Developing
i-αppli
with Star-1.0/
DoJa-5.1
Hidekazu Furukawa

Aシリーズ対応
Star iアプリ開発テキストブック

布留川英一



Developing i-αppli with Star-1.0/DoJa-5.1

Hidekazu Furukawa

Aシリーズ対応

Star iアプリ開発テキストブック

布留川英一



マイコミ
毎日コミュニケーションズ

Developing
i-αppli
with Star-1.0/
DoJa-5.1
Hidekazu Furukawa

iアプリ開発に、この1冊！

- 最新Aシリーズで採用された Star-1.0 プロファイルに完全対応。
- DoJa-5.1 プロファイルにも対応！90Xi/80Xi/70Xi シリーズの開発にも最適。

Contents

ISBN978-4-8399-3077-6
C3055 ¥3400E



定価：本体3,400円＋税



Developing
i-αppli
with Star-1.0/
DoJa-5.1

Hidekazu Furukawa

Aシリーズ対応
Star iアプリ開発テキストブック

布留川英一



Developing i-αppli with Star-1.0/DoJa-5.1

Hidekazu Furukawa

Aシリーズ対応

Star iアプリ開発テキストブック

布留川英一





A シリーズ対応

Star iアプリ開発テキストブック

布留川英一

マイコミ
毎日コミュニケーションズ

notes

- 本書で解説のある開発ツール用のプロジェクトファイル、ソースコード等各種コンテンツについては以下より入手できます。

http://book.mycom.co.jp/support/pc/star/

- iアプリ用のサイトは以下に用意しています。



Star Star-1.0プロファイル用iアプリ
http://book.mycom.co.jp/support/pc/star/i.html



DoJa DoJa-5.1プロファイル用iアプリ
http://book.mycom.co.jp/support/pc/star/d/i.html

- 本書では2009年2月現在発売されている、株式会社 NTTドコモの携帯電話（iモード対応端末）のAシリーズ（Star/DoJaプロファイル）、および90Xi、80Xi、70Xiシリーズ（DoJaプロファイル）で動くiアプリの作成方法を解説しています。書籍発刊以降に発売される新機種においての動作の保証はできません。
- 株式会社 毎日コミュニケーションズは、株式会社 NTTドコモと何らの関係もありません。
- 本書に記載の記事、製品名、URL等は2009年2月現在のものです。これらは変更される可能性があります。また記載のURLやサポートサイトは恒久的に設置するものでなく、予告無しに閉じることがあります。あらかじめご了承ください。
- 本書に記載された内容は、情報の提供のみを目的としており、本書を用いての運用は、全てお客様自身の責任と判断において行ってください。また、本書中で想定している開発環境と異なる環境において、操作および動作結果などが紙面の通りにならないことがあります。
- 本書の内容に関してなんらかの保証をするものではなく、本書の記述の実行、プログラムコードの使用により発生した損失や損害、その他いかなる事態についても、弊社および著作権者、各ソフトウェアの製作・提供者は一切の責任を負いません。あらかじめご了承ください。

introduction
はじめに

本書は、NTTドコモが策定したAシリーズ向けiアプリの新APIである**Starプロファイル**のプログラミングについて解説したものです。同時に90Xi/80Xi/70Xiシリーズ向けの**DoJaプロファイル**のプログラミングについても並行して解説していきます。これからStarプロファイル対応のiアプリを作りたいと思っている人はもちろん、作ったことはあるが新機能をあまり使いこなせてないという人も対象としています。

iアプリとは、NTTドコモの携帯電話にダウンロードして実行できるアプリケーションのことです。最近ではミニゲームやテレビゲームからの移植だけでなく、通信を利用したオンラインゲームも人気となってきました。また電子マネー関連やGPSを活用したアプリケーションなどもiアプリで提供されていることが増えています。しかしiアプリで何より面白いのは、自分で作ったゲームも実行できることです。開発ツールは無償で提供されているので、誰でもすぐにiアプリ作成をスタートすることができます。

本書は次の8章構成となっています。

1. iアプリとJava
2. iアプリ作成の基礎
3. 基本API
4. ネイティブアプリケーション連携
5. スクラッチパッドと通信
6. 3Dグラフィックス
7. Star/DoJaの独自機能
8. ゲームの作成

iアプリの作成にはプログラミング言語の**Java**を利用します。Javaの文法に関する詳しい解説は省略しているので、プログラミングがはじめてという人は、他のJavaの入門書や解説サイトに目を通してから読むことをお勧めします。

本書をより面白いゲームや便利なツールを作るための手助けにしていただければ幸いです。

最後になりますが、毎日コミュニケーションズの山口正樹さん、その他協力してくれた方々ありがとうございました。

布留川 英一

contents

Developing i-αppli with Star-1.0/DoJa-5.1

Star iアプリ開発テキストブック Aシリーズ対応

contents

how to use
本書の使い方

iアプリの開発に必要な環境：

iアプリの開発にはNTTドコモより提供されている「i-αppli Development Kit for Star-1.0」「i-αppli Development Kit for DoJa-5.1」を利用しますが、この開発ツールはMicrosoft Windows 2000/XPのみの対応となっています。**Windows Vistaには対応していません。**誌面ではWindows XPをメインに紹介しています。

本書で対象とするiモード端末機種：

本書では2009年2月現在発売されている、株式会社 NTTドコモの携帯電話（iモード対応端末）のAシリーズの**Star-1.0/DoJa-5.1プロファイル**の対応端末、および90Xi、80Xi、70Xiシリーズの**DoJa-5.1プロファイル**の対応端末で動くiアプリの作成方法を解説しています。

書籍発刊以降に発売される新機種においての動作の保証はできません。

プロジェクトファイルのダウンロードについて：

本書の執筆で作成した、各開発環境の設定ファイル/ソースコードについては、以下の毎日コミュニケーションズのサポートサイトからダウンロード可能です。

```
http://book.mycom.co.jp/support/pc/star/
```

ZIP形式でアーカイブ化していますので、ファイルを展開後「i-αppli Development Kit」の「apps」フォルダ（C:¥iDKStar1.0¥apps、もしくはC:¥iDKDoJa5.1¥apps）にコピーすると、開発に利用できます。

QRコードについて：

各記事に記載したQRコードは、その章で解説しているiアプリのダウンロードサイトへのブックマークです。実機での動作を確認したい場合にアクセスし、iアプリをダウンロードしてください。

＊ iアプリの利用は無料です（通信費用、パケット利用料金は別途かかりますのでご注意ください）。

対応機種

解説するiアプリの対応機種、およびStar-1.0、DoJa-5.1各プロファイルのようにiアプリを提供しています。

Star DoJa マーク

Star-1.0プロファイルとDoJa-5.1プロファイルのコード記述方法の違いについて、マークで示しています。

iアプリ実行例

DoJaプロファイルとの違い

本書は基本的にStar-1.0プロファイルのコード例を解説しています。DoJa-5.1プロファイルの記述方法の違いについては、コード中にⓐ～のマークで指示しています。

コード解説

コードの詳しい解説については、❶～のマークで指示しています。後の本文で詳しい解説があります。

クラス - メソッド凡例：

本書で解説しているクラスで、Star-1.0/Doja-5.1プロファイルの基本および拡張・オプションAPI共通のものについては特に記載をしていませんが、異なる場合のみ Star および DoJa のマークを入れて分類しています。

Star [StarApplicationクラス]

void started(*launchType*)

[解説] アプリケーション起動時に呼ばれる

[引数] *launchType*　　起動元種別：int型

DoJa [IApplicationクラス]

void start()

[解説] アプリケーション起動時に呼ばれる

how to use

Windows上で開発に必要となる設定について

Windows XP/2000の初期設定では、ファイル拡張子（.javaや.gifなど）が見れないようになっています。またWindows XPで開発する場合に「Thumbs.db」がiアプリに含まれないよう、以下の設定を行っておく必要があります。

コントロールパネルの［フォルダオプション］を選択してフォルダオプションダイアログを開きます。

［表示］タブを選び「詳細設定：」で、

「登録されている（ファイルの）拡張子は表示しない」

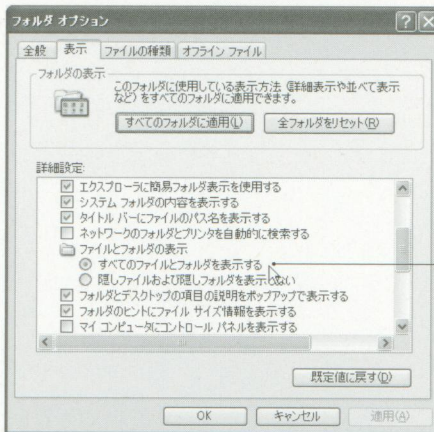
のチェックを外してください。またWindows XPの場合は

「すべてのファイルとフォルダを表示する」

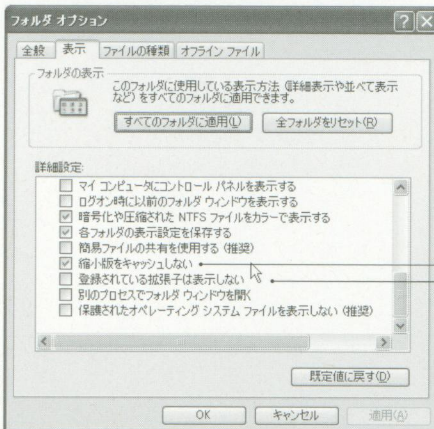
を選択し、

「縮小版をキャッシュしない」

にチェックを入れます。設定が終わったら［OK］ボタンをクリックしてください。



選択する



チェックを入れる

チェックを外す

iアプリとJava

1

1-1 iモードとiアプリ

NTTドコモが1999年から開始した**iモード**（**i-mode**）は、メールのやりとりや情報サイトを閲覧ができる、携帯電話による文字情報サービスです。

2001年に発売されたiモード対応の携帯電話**503iシリーズ**からは、ゲームや情報ツールなどのアプリケーションを携帯電話にダウンロードして使えるようになりました。このアプリケーションのことを**iアプリ**（**i-αppli**）と呼びます。

NTTドコモ以外に携帯電話を発売しているキャリア（通信事業者）には、**ソフトバンク**や**au**（**KDDI**）などが存在します。この2社もNTTドコモと同様に、iモードやiアプリに相当するサービスを提供しています。

ソフトバンクの携帯電話においては、情報サービスを**Yahoo!ケータイ**、アプリケーションを**S!アプリ**と呼びます。

auの携帯電話においては、文字情報サービスを**EZweb**、アプリケーションを**EZアプリ**と呼び、それぞれインターネット技術との親和性が高いものとなっています。

キャリア	文字情報	アプリケーション
NTTドコモ	iモード	iアプリ
ソフトバンク	Yahoo!ケータイ	S!アプリ
au	EZweb	EZアプリ

キャリア間での通話はもちろん可能ですが、アプリの互換性ありません。従って、NTTドコモの携帯電話でS!アプリを使ったり、ソフトバンクの携帯電話でiアプリを使ったりすることはできません。

▶iアプリの利点

iアプリの制作には以下のような利点が挙げられます。

・開発ツールが無償提供されている

iアプリの開発ツールは、NTTドコモやサン・マイクロシステムズのサイトから無償で提供されています。ネットに接続できるWindowsマシンとiアプリ対応の携帯電話があれば、誰でもすぐに開発をはじめられます。

・配布が容易

iアプリの配布方法は簡単で、iアプリの実行ファイルをダウンロード用HTMLと一緒に公開されたHTTPサーバ上にアップロードするだけです。検証機関によるチェックもありません。利

用者は携帯端末でダウンロード用HTMLにアクセスしてダウンロードリンクをクリックするだけで、iアプリを利用できます。

・通信機能を標準装備している

携帯電話は通信機能を標準装備しているので、ネットを活用したiアプリを作成することができます。ゲーム開発であれば全国ランキングの機能を付けることや、アップデータを配布することも容易に行えます。

▶公式アプリと勝手アプリ

携帯の世界において**公式サイト**と呼ばれているサイトは、公式メニュー（iモードではiMenu）に登録されているサイトのことです。そして、公式サイトから提供されているiアプリは**公式アプリ**と呼ばれています。公式サイトの特徴として、有料コンテンツの利用料金を電話の基本料金と一緒に支払えることが挙げられます。

勝手サイトと呼ばれているサイトは、公式メニューに登録されていないサイトのことで、主に一般ユーザが個人的に作成した携帯サイトです。同様に、勝手サイト上で配布されているiアプリのことを**勝手アプリ**と呼びます。

Column»　携帯アプリの登録サイト

携帯アプリの登録サイトには、一般ユーザが作った勝手アプリがたくさん登録されています。勝手アプリの多くは無料で公開されており、公式アプリより独創的で面白いものもたくさんあります。

携帯アプリの登録サイトの中でも特に有名なのが**アプリ★ゲット**です。「ダウンロードランキング」や「今週のおすすめ」などの各種コーナーがとても充実しており、パソコン用サイトでの閲覧も可能となっています。

・アプリ★ゲット
http://appget.com/

1-2 iアプリ対応端末

▶iアプリ対応端末のシリーズ

iアプリ対応端末としてはNTTドコモから次のようなシリーズが提供されています。

・Aシリーズ
・90Xiシリーズ
・80Xiシリーズ
・70Xiシリーズ

Aシリーズは2008年冬より発売された最新端末です。NTTドコモの販売においてはSTYLEシリーズ、PRIMEシリーズ、SMARTシリーズ、PROシリーズと分類されていますが、ユーザのライフスタイルによる分類であって、技術的な区別ではありません。

90Xiシリーズは2008年冬以前のハイスペック端末です。**70Xiシリーズ**は90Xiシリーズの廉価版で、機能を少なくして価格を安く抑えています。**80Xiシリーズ**は特殊仕様の企画端末です。また、海外メーカの端末には、iアプリ非対応の端末も存在します。

iアプリ対応端末としてはFOMA以前の50Xiシリーズも存在しますが、バージョンも古いため本書の対象外としています。

・FOMAとHSDPA

回線の種類には**FOMA**と**HSDPA**の2種類が存在します。

FOMAはNTTドコモが2001年10月からサービスを開始したIMT-2000方式による携帯電話サービスです。アナログ携帯電話を第1世代、デジタル携帯電話を第2世代とし、FOMAは**第3世代**（3G）と位置付けられています（G:Generationの意）。

HSDPAはHigh Speed Downlink Packet Accessの略で、FOMAのデータ通信を高速化した企画です。**3.5G**とも呼ばれ、FOMAの5倍以上の通信速度を実現します。

・StarとDoJa

iアプリには**Starプロファイル**と**DoJaプロファイル**の2つのプロファイルが存在し、さらにDoJaプロファイルにはいくつかのバージョンが存在します。新しいバージョンのプロファイルに対応した携帯端末で旧バージョンのアプリを使うことはできますが、旧バージョンのみに対応した携帯端末で新しいバージョンのアプリを使うことはできません。Star/DoJaに対応しているかどうかは個々の端末ごとに異なりますが、特にStarプロファイルはAシリーズ以降の対応です。

iアプリ対応端末のシリーズごとの「プロファイル」と「回線の種類」は次の表の通りです。

シリーズ	プロファイル	回線の種類
Aシリーズ	Star-1.0/DoJa-5.1	FOMA/HSDPA
906iシリーズ	DoJa-5.1	FOMA/HSDPA
905iシリーズ	DoJa-5.1	FOMA/HSDPA
904iシリーズ	DoJa-5.0	FOMA/HSDPA
903iシリーズ	DoJa-5.0	FOMA/HSDPA
902iシリーズ	DoJa-4.1	FOMA
901iシリーズ	DoJa-4.1	FOMA
900iシリーズ	DoJa-3.5	FOMA
80Xiシリーズ	機種依存	FOMA
706iシリーズ	DoJa-5.1LE	FOMA/HSDPA
705iシリーズ	DoJa-5.0LE/5.1LE	FOMA/HSDPA
704iシリーズ	DoJa-4.1LE/5.0LE	FOMA/HSDPA
703iシリーズ	DoJa-4.1LE/5.0LE	FOMA
702iシリーズ	DoJa-4.0LE/4.1LE	FOMA
701iシリーズ	DoJa-4.0LE	FOMA
700iシリーズ	DoJa-4.0LE	FOMA

発売された機種がどのプロファイルに対応しているかは、以下のサイトを参照してください。端末ごとの具体的な情報についても確認できます。

- **作ろうiモードコンテンツ：iアプリ**
  http://www.nttdocomo.co.jp/service/imode/make/content/spec/iappli/index.html

本書の付録a-1（346ページ）にもStar-1.0/DoJa-5.1プロファイル対応端末の一覧を掲載しましたので、あわせて確認してください。

▶Aシリーズ

2009年2月現在、発売および発表されているAシリーズの端末と対応するプロファイルは以下の通りです。

プロファイル名	機種名
Star-1.0	F-01A、F-03A、N-01A、N-02A、N-04A、P-01A、P-02A、SH-01A、SH-03A、SH-04A
DoJa-5.1	F-06A、N-03A、P-03A、P-04A、P-05A、SH-02A
DoJa-5.1LE	F-02A、F-04A
DoJa-5.0LE	F-05A
DoJa-4.1LE	L-01A
（なし）	HT-01A、HT-02A、BlackBerry Bold

※（なし）の機種ではiアプリが利用できません。

Column» Aシリーズの構成

2009年2月現在発売されているAシリーズの構成を整理すると、以下のようになります。

STYLEシリーズ：ファッショナブルケータイ
Star-1.0：N-02A、P-02A
DoJa-5.1：N-03A、P-03A、SH-02A
DoJa-5.1LE：F-02A
PRIMEシリーズ：新世代エンタテインメントケータイ
Star-1.0：F-01A、F-03A、N-01A、P-01A、SH-01A、SH-03A
DoJa-4.1LE：L-01A
SMARTシリーズ：インテリジェントケータイ
Star-1.0：N-04A
DoJa-5.1：P-04A、P-05A
DoJa-5.1LE：F-04A
PROシリーズ：デジタルマスターケータイ
Star-1.0：SH-04A
（なし）：HT-01A、HT-02A、BlackBerry Bold
コンセプトモデル
DoJa-5.1：F-06A（法人向端末）
キッズケータイ
DoJa-5.0LE：F-05A

▶90Xiシリーズ

2009年2月現在、発売されている90Xiシリーズの端末と対応するプロファイルは以下の通りです。

プロファイル名	機種名
DoJa-5.1	D905i、F905i/Biz、N905i/μ/Biz、P905i/TV、SH905i/TV、SO905i/CS、P906i、SO906i、SH906i/TV、N906i、F906i、N906i/μ/L
DoJa-5.0	SH903i/TV、P903i/TV/X（P903iX HIGH-SPEED）、N903i、D903i/TV、F903i/BSC/X（F903iX HIGH-SPEED）、SO903i/TV、SH904i、N904i、F904i、D904i、P904i
DoJa-4.1	D902i/S、F902i/S、N902i/S/L/X（N902iX HIGH-SPEED）、P902i/S、SH902i/S/SL（DOLCE SL）、SO902i/S/WP+
DoJa-4.1LE	N902iL
DoJa-4.0	D901i/S、F901iC/S、N901iC/S、P901i/S/TV、SH901iC/S
DoJa-3.5	D900i、F900i/T/C、N900i/S/L/G、P900i/V、SH900i

▶80Xiシリーズ

2009年2月現在、発売されている80Xiシリーズの端末と対応するプロファイルは以下の通りです。

プロファイル名	機種名
DoJa-5.1LE	F884i（らくらくホン プレミアム）、F884iES（らくらくホンV）
DoJa-5.0LE	F801i（キッズケータイ）
DoJa-4.1LE	D800iDS、D851iWM（Music Porter X）
DoJa-4.0LE	P851i（prosolid II）、SA800i（キッズケータイ）
DoJa-3.5LE	L852i（PRADA Phone by LG）

▶70Xiシリーズ

2009年2月現在、発売されている70Xiシリーズの端末と対応するプロファイルは以下の通りです。

プロファイル名	機種名
DoJa-5.1 LE	P705i/μ/CL（PROSOLIDμ）、SO705i、F706i、P706iμ/e
DoJa-5.1	N705i/μ、N706i/e/II、SO706i、SH706i/e/w
DoJa-5.0	SH703i、SO703i、SH704i、SO704i、SH705i/II、SH706ie
DoJa-5.0LE	D703i、F703i、N703iD/μ、P703i、D704i、F704i、N704iμ、P704i、D705i/μ、F705i
DoJa-4.1LE	D702i/BCL/F、F702iD/F、P702i/D、SA702i、SH702iD/S、SO702i、P703iμ、P704iμ
DoJa-4.0LE	F700i/S、N700i、P700i、SA700iS、SH700i/S、D701i/WM（Music Porter II）、N701i/ECO、P701iD、N702iD、N702iS、M702iS、M702iG
DoJa-3.5LE	L704i、L705i/X、L706ie

Column» 携帯電話の名前

NTTドコモの端末の名前は型番なので、はじめて見た時はアルファベットと数字ばかりで戸惑うかもしれません。しかし命名規則さえ覚えれば理解しやすくなります。

・Aシリーズ以降

Aシリーズ以降の命名規則は以下のようになります。

（例） F-01A
[メーカーのイニシャル]-[通し番号][登場年度]

[登場年度]は、2008年冬モデルから2009年夏モデルまでは「**A**」、次の冬モデルから翌夏モデルは「**B**」になります。[**メーカーのイニシャル**]は次の通りです。

イニシャル	メーカー名
D	三菱電機
F	富士通
H	HTC
L	LGエレクトロニクス・ジャパン
M	モトローラ
NM	ノキア・ジャパン
N	NEC
P	パナソニック モバイルコミュニケーションズ
SH	シャープ
SO	ソニーエリクソン

たとえば、富士通（F）の1台目（01）の2008年冬モデル（A）は**F-01A**となります。

・Aシリーズ以前

Aシリーズ以前の命名規則は以下のようになります。

（例） P906i
[メーカーのイニシャル][シリーズ名]

「iS」等、名前の後ろに「**S**」が付いているのが、基本機能は数字のシリーズ名そのままに若干パワーアップさせた端末（セカンドモデル）で、「**V**」が付いているのが高機能な端末（ハイエンドモデル）です。タッチパネル対応端末に「**T**」、FeliCa対応端末に「**C**」など、機能に応じて付くこともあります。

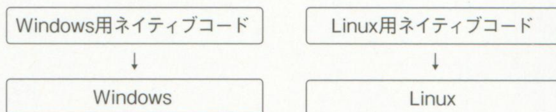
たとえば、パナソニック（P）の906iシリーズの端末は**P906i**となります。

1-3 JavaとJavaME

▶Java

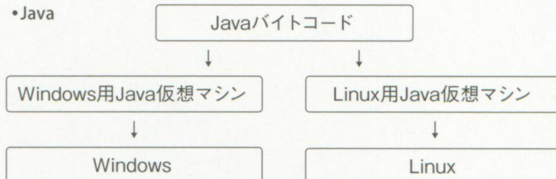
iアプリではプログラミング言語として**Java**を利用します。Javaは、サン・マイクロシステムズの開発したプログラミング言語です。ネイティブコードに変換して実行するプログラミング言語とは異なり、**Javaバイトコード**と呼ばれる中間言語に一度変換し、実行時に**Java仮想マシン（JVM：Java Virtual Machine）**と呼ばれるソフトウェアによって、そのプラットフォームで実行可能な形式に変換して実行するという方法をとっています。これによって、異なる端末で同じプログラムを実行しやすくしています。

・従来のプログラミング言語



プラットフォームごとに異なるネイティブコードを作る必要がある

・Java



Javaバイトコードはjava仮想マシンを介してさまざまなプラットフォーム上で動く

▶Java実行環境

Javaは、デスクトップはもちろん、サーバや携帯端末などさまざまな用途で使われており、機能も膨大です。そのため、Java実行環境は3つのエディションに分けられています。**JavaSE**はデスクトップ環境で使われるJava実行環境です。**JavaEE**はJavaSEにWebアプリケーション作成のための機能を追加したものです。**JavaME**は携帯端末に特化し、機能を絞ってコンパクトにしたものです。

- **JavaSE：デスクトップ向け**
  http://java.sun.com/javase/

・JavaEE：サーバ向け
http://java.sun.com/javaee/

・JavaME：携帯端末向け
http://java.sun.com/javame/

▶JavaME

JavaSEには**JavaSE SDK**、JavaEEにはJavaEE SDKという開発キットが、サン・マイクロシステムズから提供されていますが、JavaMEには特定の開発キットがありません。これはJavaMEが対象とする組み込み機器は、処理能力が異なるのはもちろん、インターフェイスも多種多様なため、1つの開発キットとして提供することができないからです。JavaMEでは、機器に応じて次で説明する**コンフィギュレーション**と**プロファイル**を組み合わせて使うことになります。

・コンフィギュレーション

JavaMEでは処理能力の低い機器でもJavaプログラムが動くように、従来のJavaSEから機能を絞りこんだ**サブセット**を定義しています。この定義のことを**コンフィギュレーション**（**configuration**）と呼びます。2009年現在、次の2つのコンフィギュレーションが存在します。

JavaME CDCは、カーナビ、セットトップ・ボックスなどの中程度の処理能力を持つ端末を対象としているコンフィギュレーションで、Java仮想マシンはJavaSEで使われているものと同じものを利用します。

・JavaME CDC（JavaME Connected Device Configuration）
http://java.sun.com/products/cdc/

JavaME CLDCは、処理能力やメモリに制限のある携帯端末を対象としているコンフィギュレーションです。サポートするクラスの数を必要最低限に絞って、クラスライブラリ自体のサイズを減らして、Java仮想マシンが実行時に使用するメモリを小さくしています。

・JavaME CLDC（JavaME Connected, Limited Device Configuration）
http://java.sun.com/products/cldc/

CLDC仕様のJavaプログラムを動作させるためのJava仮想マシンは、キロ（K）のオーダーのメモリで動作することから**KVM**と呼ばれています。iアプリの開発には、このコンフィギュレーションを使います。

・プロファイル

組み込み機器は、タッチパネルや非接触ICなどそれぞれ特有の機能を持つため、それに対応できるように用途に応じた特殊なAPI（Application Programming Interface）が必要になります。このAPIのことを**プロファイル（Profile）**と呼びます。2009年現在、JavaME CLDCには次の3つのプロファイルがあります。

MIDPは、世界標準の携帯端末向けプロファイルです。ソフトバンクやauのJava実行環境は、このMIDPがベースとなっています。

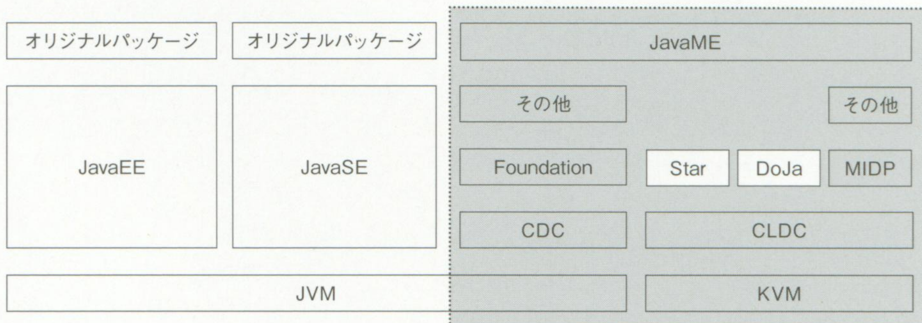
- **MIDP（Mobile Information Device Profile）**
  http://java.sun.com/products/midp/

Starプロファイルと**DoJa（Docomo Java）プロファイル**はNTTドコモの定めるiアプリ用のプロファイルです。MIDPとは別のプロファイルを用意することにより、iモード端末の機能をより活かせるようにしました。

- **iアプリ（Starプロファイル、DoJaプロファイル）**
  http://www.nttdocomo.co.jp/service/imode/make/content/iappli/

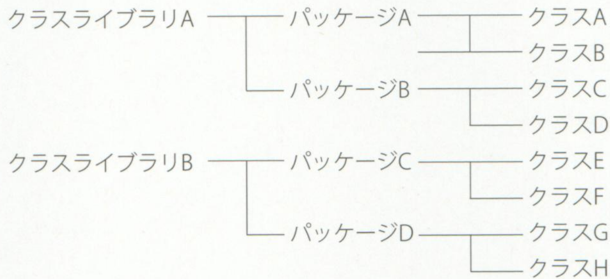
503iシリーズから906iシリーズまではDoJaプロファイル、2008年冬に発売されたAシリーズからはStarプロファイルが標準プロファイルとなります。ただし、前述のようにAシリーズであってもStarプロファイルに未対応の端末もあります。また、現在のStarプロファイル対応端末はDoJaプロファイルにも対応しています。

以上、この節で解説した用語を図に整理すると以下のようになります。



1-4 iアプリのクラスライブラリ

特定の機能を持ったプログラムを部品化したもののことを**クラス**（class）と呼びます。関連するクラスをまとめたもののことを**パッケージ**（package）、関連するパッケージをまとめたもののことを**クラスライブラリ**（class library）と呼びます。



iアプリで利用できるクラスライブラリには、次の3種類があります。

- **JavaME CLDCのAPI**
- **iアプリ基本API**
- **iアプリオプション・拡張API**

▶JavaME CLDCのAPI

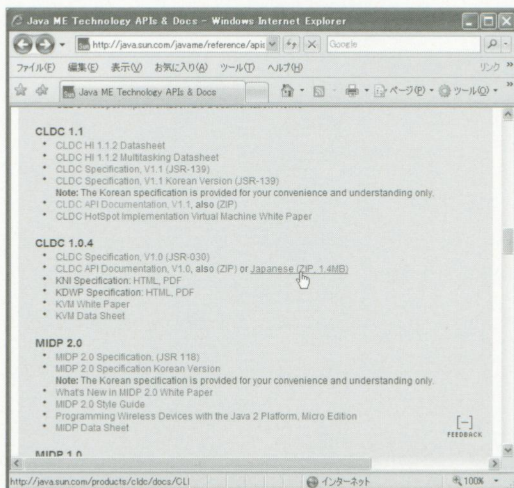
JavaME CLDCのAPIは、処理能力やメモリなどの制限のある携帯端末を対象としている汎用的なAPIで、iアプリだけでなく、ソフトバンクやauのアプリでも利用されています。このAPIは、次の5種類のパッケージを持っています。

パッケージ名	説明
java.lang	Java言語の基本クラス
java.lang.ref	ウィークリファレンスのサポート
java.io	データの入出力
java.util	ユーティリティ
javax.microedition.io	データの入出力先への接続

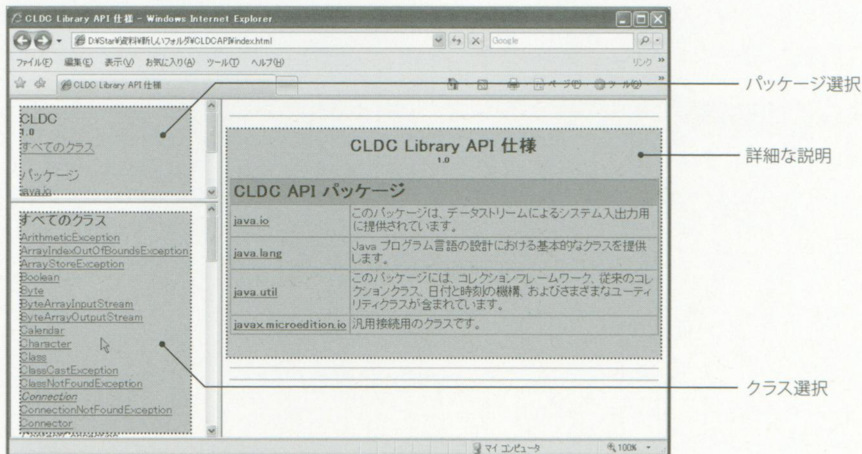
JavaME CLDCのAPIにおいて具体的にどのようなクラスがあるのか、またそのクラスにはどのようなメソッドが用意されているのかを調べるには、**APIリファレンス**と呼ばれるドキュメントを活用します。サン・マイクロシステムズのサイトにあるJavaMEドキュメントのページか

ら、「CLDC API Documentation, V1.0」「Japanese (ZIP, 1.4MB)」を選択してダウンロードしてください。バージョンはCLDC1.0（CLDC1.0.4）になります。

- **Java ME Technology APIs & Docs**
  http://java.sun.com/javame/reference/apis.jsp



ダウンロードしたファイル**CLDC1.0_ja_docs.zip**はZIP形式で提供されており、展開すると「CLDCAPIhtml.zip」ができます。それをさらに解凍すると「CLDCAPI」というフォルダができ、フォルダ内の「index.html」をダブルクリックすることで、WebブラウザでAPIリファレンスを閲覧することができます。

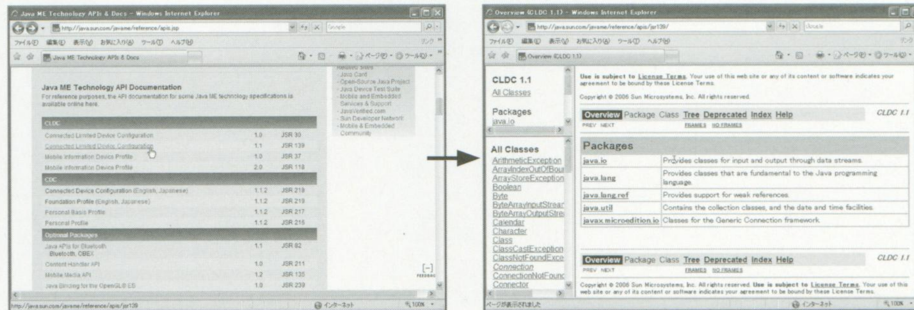


APIリファレンスは3つのフレームに分割されています。

左上のフレームで**パッケージ名**を選択すると、左下のフレームに指定したパッケージに属する**クラス**が表示されます。次に、クラスを選択すると右のフレームにクラスに属する**メソッド**が表示されます。各メソッドにはリンクが張られており、メソッド名をクリックするとより詳しい情報が表示されます。

また英語表記となりますが、最新のCLDC 1.1のAPIリファレンスをオンラインで閲覧することも可能です。

- CLDC 1.1
  http://java.sun.com/javame/reference/apis/jsr139/



▶iアプリ基本API

iアプリ基本APIは、iアプリに特化した基本的な処理を行うためのAPIです。StarとDoJaの2つのプロファイルがあり、パッケージ名も異なります。

iアプリ基本APIの主なパッケージは次の通りです。

Starパッケージ名	DoJaパッケージ名	説明
com.docomostar.ui	com.nttdocomo.ui	ユーザインタフェース
com.docomostar.io	com.nttdocomo.io	通信
com.docomostar.net	com.nttdocomo.net	MIME形式のエンコード/デコード
com.docomostar.util	com.nttdocomo.util	ユーティリティ
com.docomostar.lang	com.nttdocomo.lang	例外
com.docomostar.system	com.nttdocomo.system	ネイティブアプリケーション連携
com.docomostar.ui.graphics3d	com.nttdocomo.ui.graphics3d	3Dグラフィックス
com.docomostar.device	com.nttdocomo.device	カメラ/バーコードリーダ等
com.docomostar.device.felica	com.nttdocomo.device.felica	FeliCa
com.docomostar.device.location	com.nttdocomo.device.location	GPS
com.docomostar.fs.sd	com.nttdocomo.fs.sd	SDカード

chapter
1
2
3
4
5
6
7
8
a

基本APIだからといって、全ての端末で利用できるとは限りません。バージョンによる制限や、利用できるかどうかが機種依存となるクラスやメソッドも存在します。

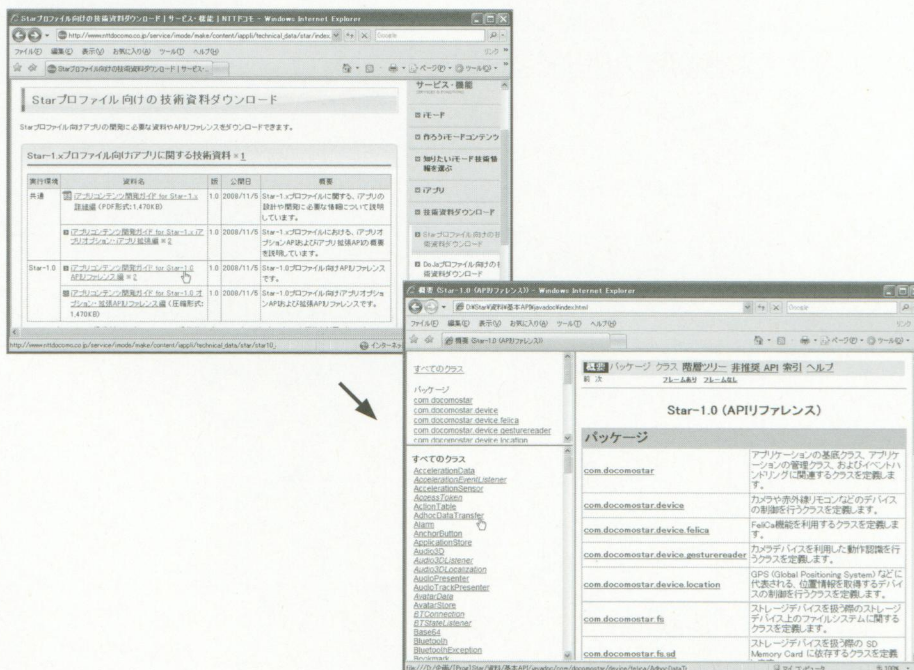
iアプリ基本APIのAPIリファレンスは、NTTドコモのサイト「技術資料ダウンロード」で入手できます。

- **技術資料ダウンロード**
  http://www.nttdocomo.co.jp/service/imode/make/content/iappli/technical_data/

- **iアプリコンテンツ開発ガイド for Star-1.0 APIリファレンス編**
  http://www.nttdocomo.co.jp/service/imode/make/content/iappli/technical_data/star/star10_apiref/index.html

- **iアプリコンテンツ開発ガイド for DoJa-5.1 APIリファレンス編**
  http://www.nttdocomo.co.jp/service/imode/make/content/iappli/technical_data/doja/doja51_apiref/index.html

Star-1.0 APIリファレンスの場合、**jguideforstar1_x_apiref_081105.zip**というファイルを展開すると「基本API」フォルダが生成されます。「javadoc」フォルダ内にあるindex.htmlを表示することで、WebブラウザでAPIリファレンスを閲覧することができます。



▶iアプリオプション・拡張API

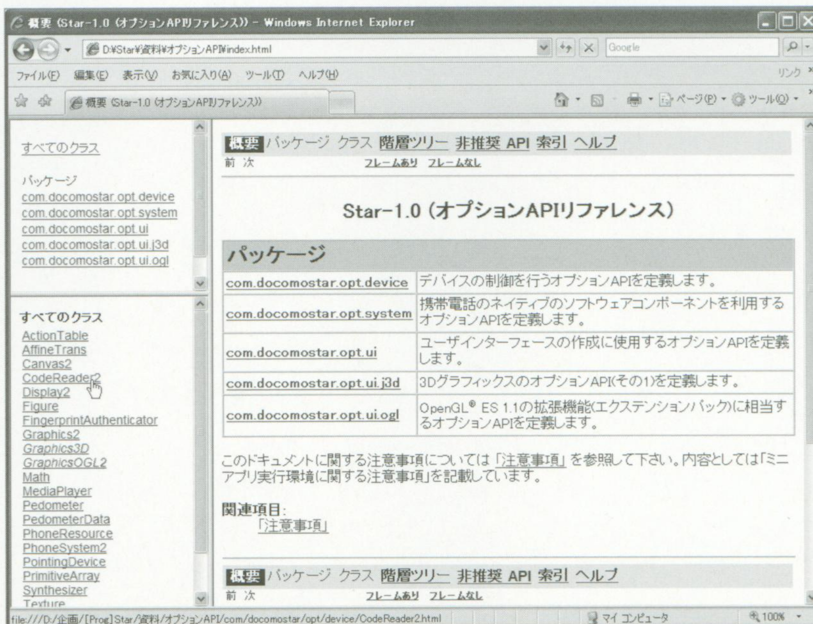
iアプリオプション・拡張APIは、機能を搭載するかどうかの判断が各端末のメーカーに委ねられているAPIで、共通の仕様が規定されています。このAPIによって、タッチパネルや非接触ICなど、特定の機種でしか使えない機能が使えるようになります。

iアプリオプション・拡張APIのAPIリファレンスはNTTドコモのサイト「技術資料ダウンロード」で入手できます。

- **iアプリコンテンツ開発ガイド for Star-1.0 オプション・拡張APIリファレンス編**
  http://www.nttdocomo.co.jp/binary/archive/service/imode/make/content/iappli/technical_data/star/jguideforstar1_x_apiref_opt_081105.zip

- **iアプリコンテンツ開発ガイド for DoJa-5.1 オプション・拡張APIリファレンス編**
  http://www.nttdocomo.co.jp/binary/archive/service/imode/make/content/iappli/technical_data/doja/jguidefordoja5_x_apiref_opt_081024.zip

「Star-1.0オプション・拡張APIリファレンス」の場合、**jguideforstar1_x_apiref_opt_081105.zip**というファイルを展開すると「オプションAPI」フォルダが生成されます。「javadoc」フォルダ内にあるindex.htmlを表示することで、WebブラウザでAPIリファレンスを閲覧することができます。



Column» iアプリDX

iアプリDXは以下の機能の利用を許可されたアプリのことです。

- **iアプリオンライン/iアプリコール**
- **端末内のユーザ情報（電話帳や発着信履歴）へのアクセス**
- **ダウンロード元サーバ以外のサーバとの通信**
- **OpenGL ES**

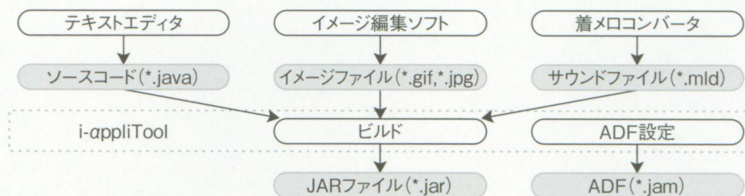
iアプリDXを配信できるのは公式アプリを提供しているコンテンツプロバイダのみで、一般ユーザは作ることはできません。一般ユーザに解放してしまうとセキュリティ的に危険な機能は、iアプリDXのみの機能として制限されています。iアプリDXはNTTドコモから提供されているドキュメントでは**トラステッドiアプリ**とも呼ばれています。

iアプリ作成の基礎

2-1 iアプリ作成とダウンロードの流れ

▶iアプリ作成の流れ

iアプリ作成の流れをまとめると次のようになります。



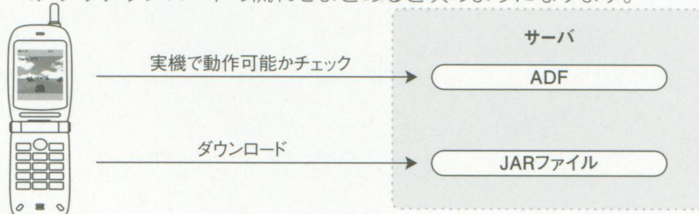
テキストエディタで**Javaのソースコード**（.java）を記述し、イメージを表示する場合は**イメージファイル**（.gifもしくは.jpg）、サウンドを再生する場合は**サウンドファイル**（.mld）を用意します。

それらのコンテンツを**iαppliTool**（iαppli Development Kit）で、iアプリの実行ファイルである**JARファイル**（.jar）に変換します。この処理を**ビルド**と呼びます。

次に、iαppliToolの「ADF設定」を使って、iアプリの属性ファイルである**ADF**（.jam）を作成します。ADFには名前やファイルサイズ、最終更新日などを記載します。

▶iアプリダウンロードの流れ

iアプリダウンロードの流れをまとめると次のようになります。



iアプリ対応端末でダウンロード用HTMLにアクセスし、ダウンロードリンクをクリックすると、まずはじめにiアプリの属性を記載したADFを先にダウンロードします。ADFをチェックして、そのiアプリが実機で動作可能かチェックし、動作可能の時はJARファイルをダウンロードします。

2-2 開発ツールの準備

iアプリを作るために必要な開発ツールは以下の2つです。

- Java 2 SDK, Standard Edition 1.4（JDK1.4）
- iαppli Development Kit

それぞれデフォルトの設定でインストールを行うとルートドライブの「¥j2sdk1.4.2_19」フォルダと「iDKStar1.0」（「iDKDoJa5.1」）フォルダにインストールされます。

▶Java 2 SDK, Standard Edition 1.4

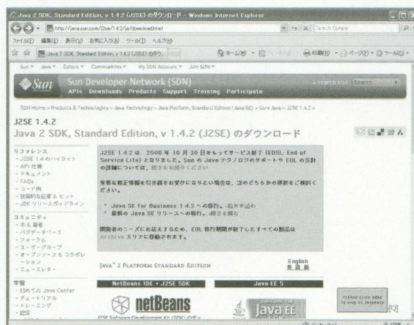
Java 2 SDK, Standard Edition 1.4（**JDK1.4**）は、パソコン上で動くJavaアプリケーションを作成するための開発キットです。サン・マイクロシステムズのサイトからダウンロードできます。**iαppli Development Kit**を実行するのに必要なため、JDK1.4を先にインストールしてください。

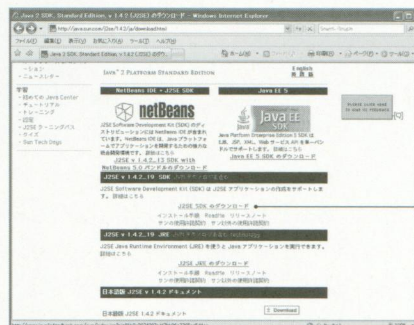
JDKのバージョンは「1.4」です。最新のJDK（1.5、1.6）でも動きますが、iαppli Development Kitのメニュー［設定］－［sun.tools.javac.Mainを使用する］にチェック入れる必要があり、コンパイル時に「推奨されません」といった警告も表示されます。

JDK1.4のダウンロード

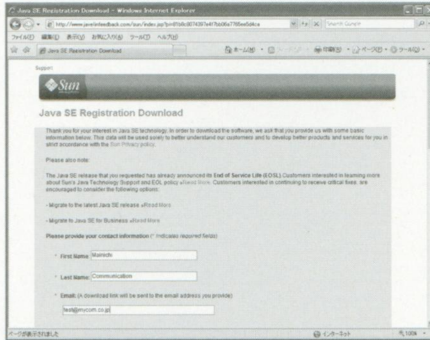
① サン・マイクロシステムズのJDK1.4のダウンロードサイトを開き、「J2SE SDKのダウンロード」をクリックします。

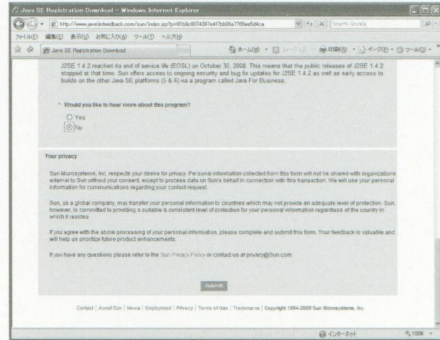
- **ダウンロード Java 2 SDK, Standard Edition, v 1.4.2（J2SE）**
  http://java.sun.com/j2se/1.4.2/ja/download.html



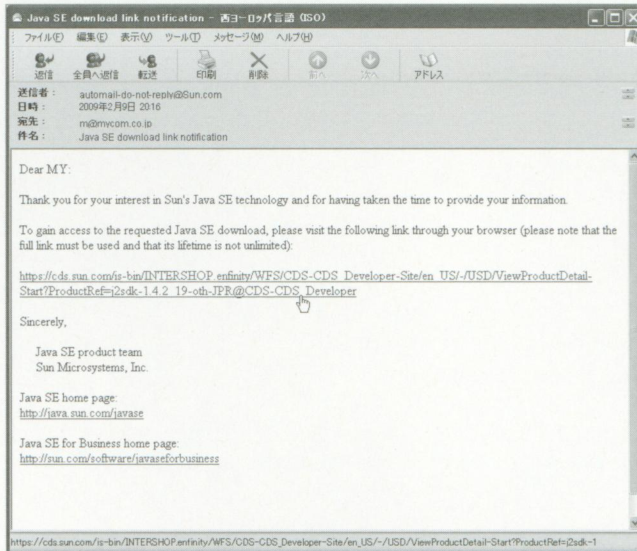


② JDKをダウンロードするには、登録が必要となります。ユーザ名やメールアドレスを尋ねるフォームが表示されますので、入力し、最後［Submit］ボタンをクリックします。

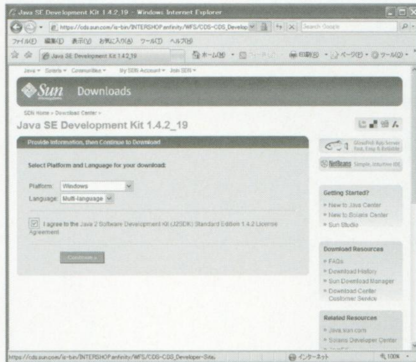


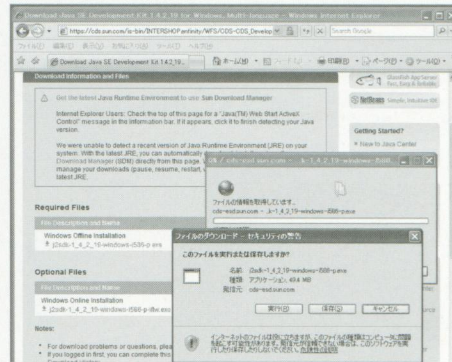


③ 登録されたメールアドレスに、ファイルのダウンロードのURLが送られてきますので、そのサイトへアクセスしてください。



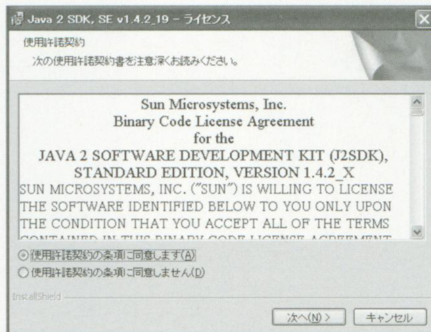
④ 開かれたページで「Platform:」は「Windows」、「Language」に「Multi-language」を選択し、ライセンスの記述を確認し「I agree to ～」にチェックを入れ［Continue>>］ボタンをクリックするとダウンロードのページに移動しますので、「j2sdk-1_4_2_19-windows-i586-p.exe」をクリックし、インストールファイルを保存してください。

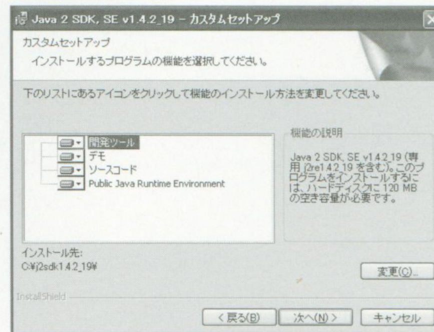




JDK1.4のインストール

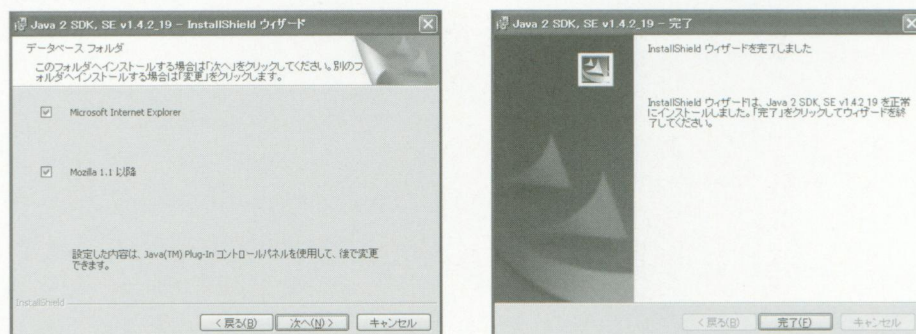
① ダウンロードしたファイル「j2sdk-1_4_2_19-windows-i586-p.exe」をダブルクリックします。ライセンスの記述を確認「～に同意します」を選択し、［次へ>］ボタンをクリックします。次の画面でインストールするコンポーネントとインストール先を選択し［次へ>］ボタンをクリックします。これはデフォルトのままでもかまいません。



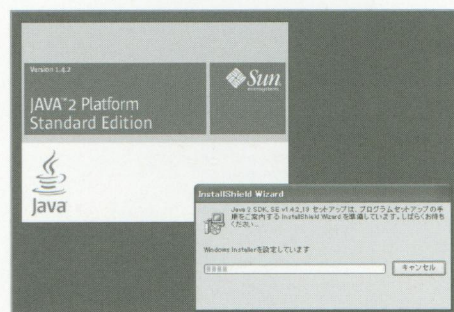


開発ツール	JDKのライブラリと実行ファイル（必須）
デモ	Javaのデモを実行するアプレットとアプリケーション
ソースコード	JDKのライブラリのソースコード
Public Java Runtime Environment	Javaの実行環境（必須）

② インストールする「Java Plug-In」を選択し（デフォルトのままでOK）、［次へ>］ボタンをクリックします。



③ するとインストールが開始され、自動的に終了します。



▶iαppli Development Kit

iαppli Development Kitは、実際にiアプリを作るための開発キットです。ボタン1つで**ビルド**や、**エミュレータでのiアプリ実行**を行えます。

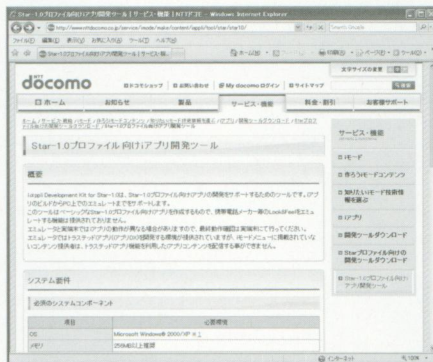
iαppli Development Kitには**Star用**と**DoJa用**のものがあります。使い方はほぼ同じですが、作成するプロファイルに応じて使い分ける必要があります。DoJa用のiαppli Development Kitには複数のバージョンが存在しますが、**最新の5.1**を利用するとよいでしょう。

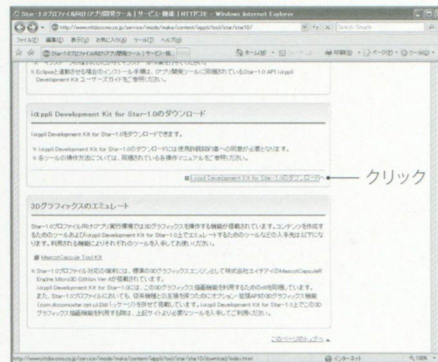
iαppli Development Kitのダウンロード

① NTTドコモのiαppli Development Kitのダウンロードサイトを開き、（Star-1.0用の開発ツールを入手する場合）「iαppli Development Kit for Star-1.0のダウンロード」のリンクをクリックしてください。

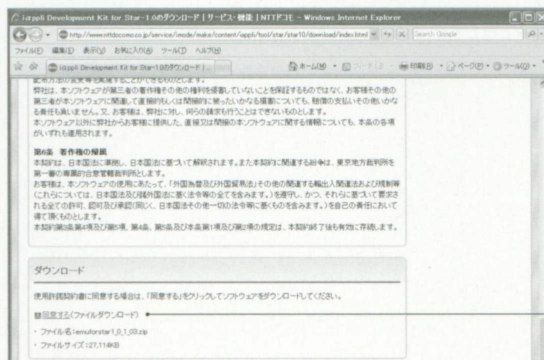
- **Star-1.0プロファイル向けiアプリ開発ツール**
  http://www.nttdocomo.co.jp/service/imode/make/content/iappli/tool/star/star10/

- **DoJa-5.1プロファイル向けiアプリ開発ツール**
  http://www.nttdocomo.co.jp/service/imode/make/content/iappli/tool/doja/doja51/



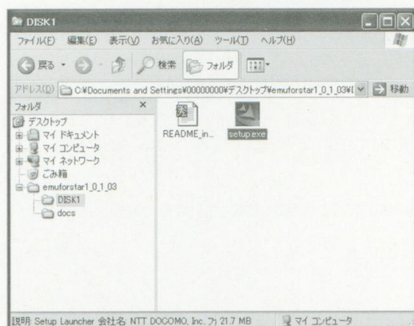


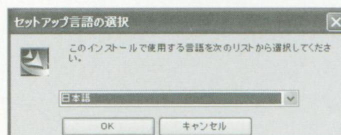
② 次のページで使用許諾契約書を読み「同意する」のリンクをクリックするとファイルダイアログが開くので、ダウンロードして「emuforstar1_0_1_03.zip」を保存、入手してください。DoJa-5.1用のツールの場合は「emufordoja5_1_3_00.zip」です。



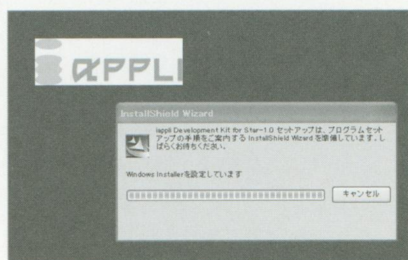
iαppli Development Kitのインストール

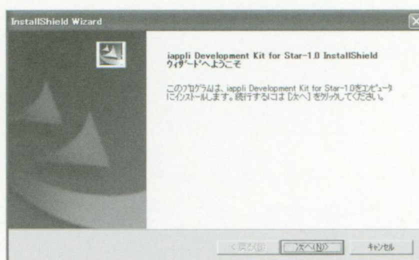
① ダウンロードしたZIPファイル「emuforstar1_0_1_03.zip（emufordoja5_1_3_00.zip）」を展開してください。「doc」フォルダと「Disk1」フォルダができますので、「Disk1」フォルダにある「setup.exe」をダブルクリックし、インストーラを起動してください。「セットアップ言語の選択」ダイアログで「日本語」を選択し、[OK]ボタンをクリックします。



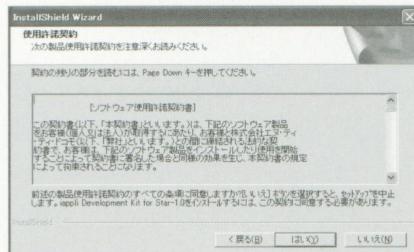


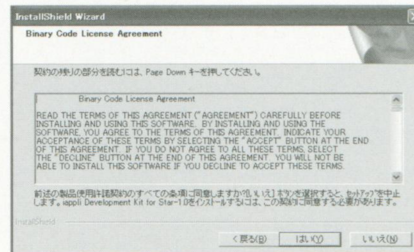
② インストーラが起動し「ようこそ」の画面が出るので、そのまま［次へ>］ボタンをクリックしてください。



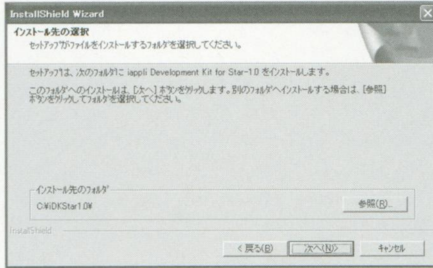


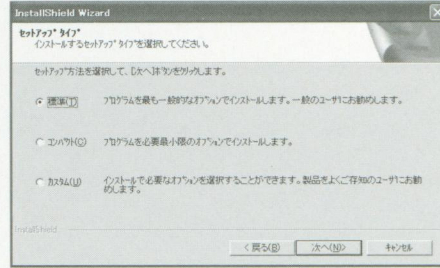
③ 次に日本語ライセンスの記述を確認し、［はい］ボタンをクリックします。続いて英語ライセンスの記述を確認し、［はい］ボタンをクリックしインストールを続けます。



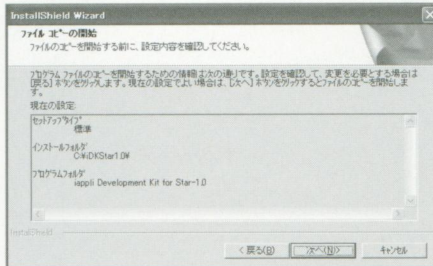


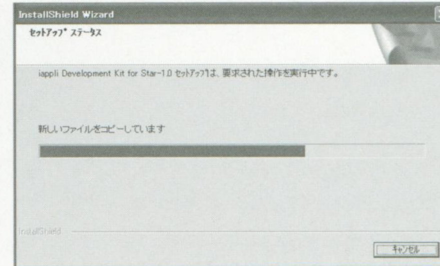
④ インストール先を選択し（デフォルトのままでOK）、[次へ>] ボタンをクリックします。次にセットアップタイプとして「標準」を選択し、[次へ>] ボタンをクリックしてください。iαppli Development Kitのデフォルトのインストール先はStar-1.0版の場合は、システムドライブの「¥iDKStar1.0」フォルダ、DoJa-5.1版の場合は「¥iDKDoJa5.1」フォルダになります。



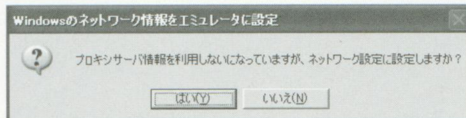


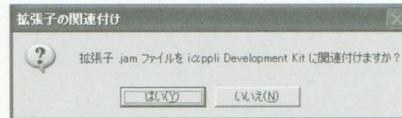
⑤ 設定を確認して、[次へ>] ボタンをクリックするとインストールが始まります。



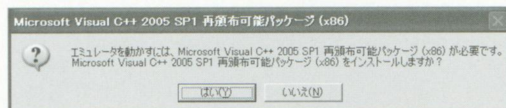


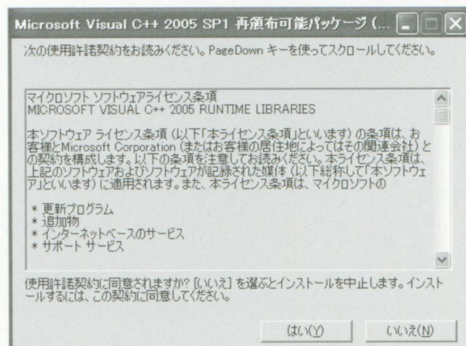
⑥ 次にネットワーク環境について尋ねられるので、[はい] ボタンをクリックします。これはエミュレータがネットワークを利用する際にプロキシサーバを利用するかどうかの設定です。次に「拡張子 .jam」のファイルを関連付けるかどうかを尋ねられますので、[はい] ボタンをクリックします。



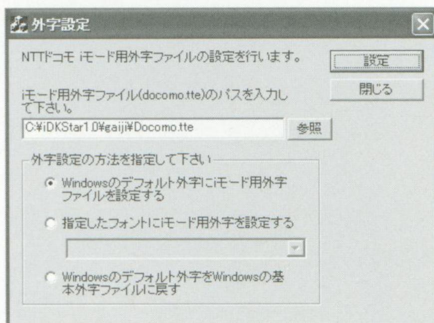


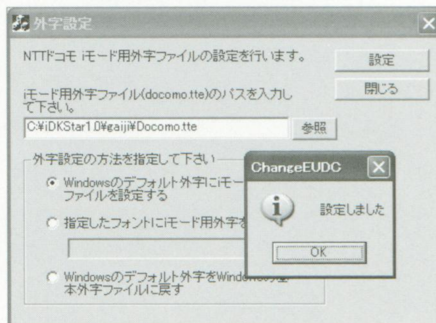
⑦ Star-1.0用のiαppliToolkitのインストール時に「Microsoft Visual C++ 2005 SP1 再頒布パッケージ」のインストール確認ダイアログが表示されますので、［はい］ボタンをクリックしインストールを行ってください。





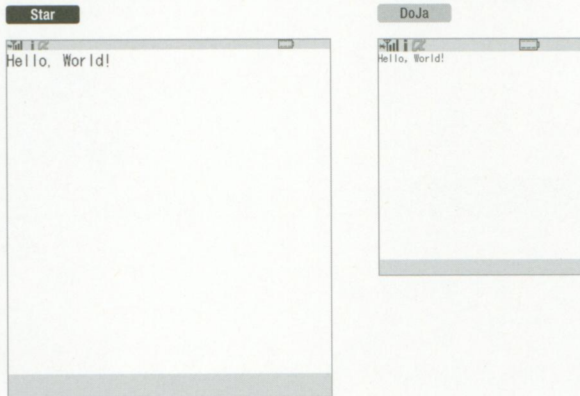
⑧ インストール完了時に、**iモードの絵文字の設定**を行う「外字設定」ダイアログが表示されます。ここでは［設定］ボタンをクリックすると、「設定しました」と表示されるので［OK］ボタンをクリックし、元のダイアログで［閉じる］ボタンをクリックしてインストールを終えてください。





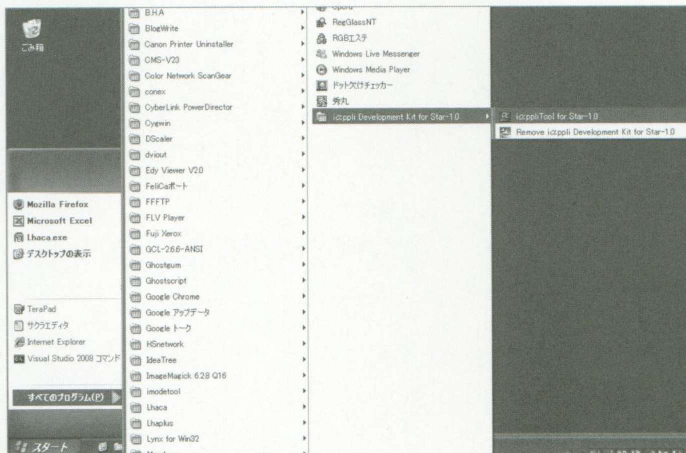
2-3 はじめてのiアプリ作成

開発ツールの準備ができたら、画面に「**Hello,World!**」と表示するプログラムを作りましょう。

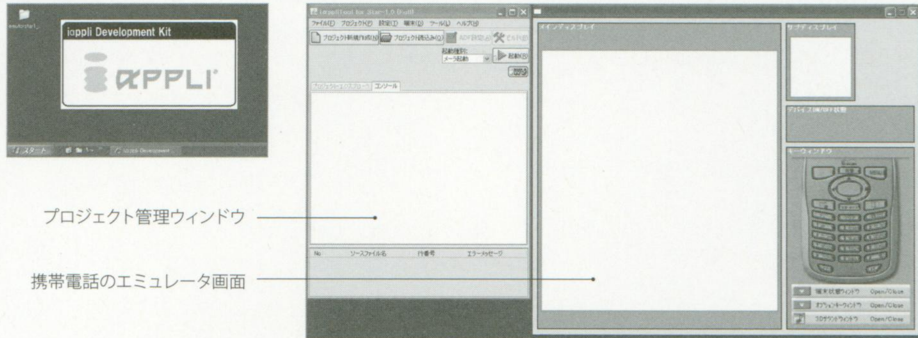


▶iαppliToolの起動

iαppliToolはiαppli Development Kitの核となるツールです。iαppli Development Kitのインストール後、Windowsのスタートメニューより［(すべての) プログラム］-［iαppli Development Kit for Star-1.0］-［i-αppliTool for Star-1.0］(DoJaの場合［iαppli Development Kit for DoJa-5.1(FOMA)］-［i-αppliTool for for DoJa-5.1(FOMA)］)が追加されているので、選択して起動します。



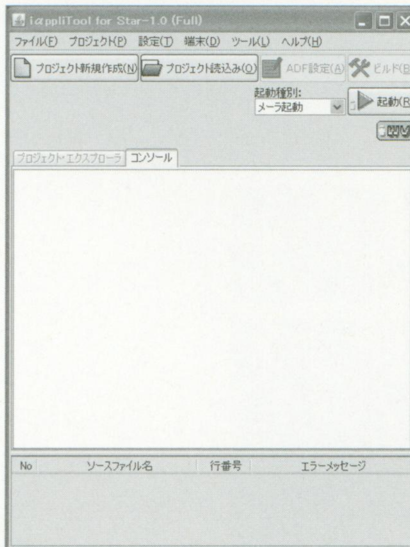
起動すると、**プロジェクト管理ウィンドウ**と、**携帯電話のエミュレータの画面**が表示されます。



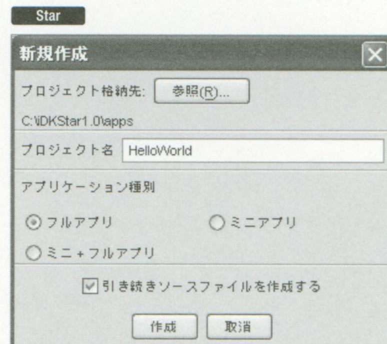
▶プロジェクトの新規作成

プロジェクトというのは、iアプリの開発に必要なファイルをiαppliToolが管理するための単位です。

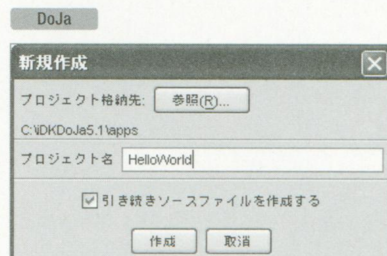
iαppliToolの［プロジェクト新規作成］ボタンをクリックすると、「新規作成」ダイアログが開くので、プロジェクト名に"HelloWorld"と入力してください。



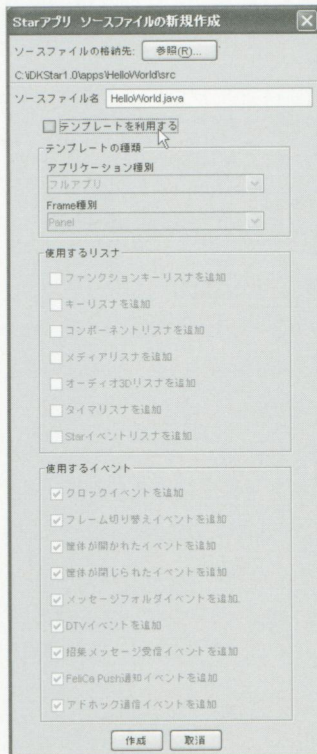
Star



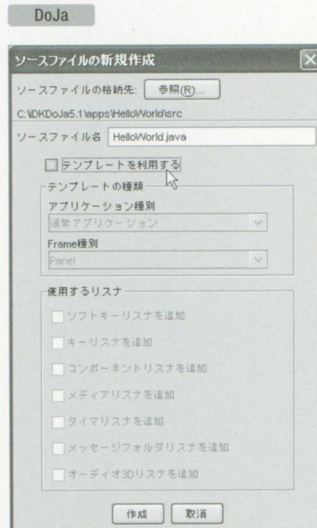
DoJa



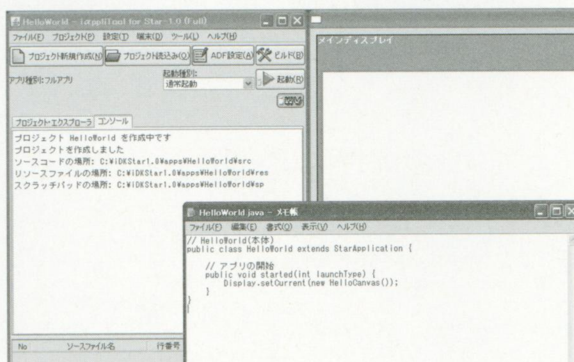
次にソースファイルの新規作成ダイアログが開くので、「テンプレートを利用する」のチェックを外し、［作成］ボタンをクリックします。



DoJa

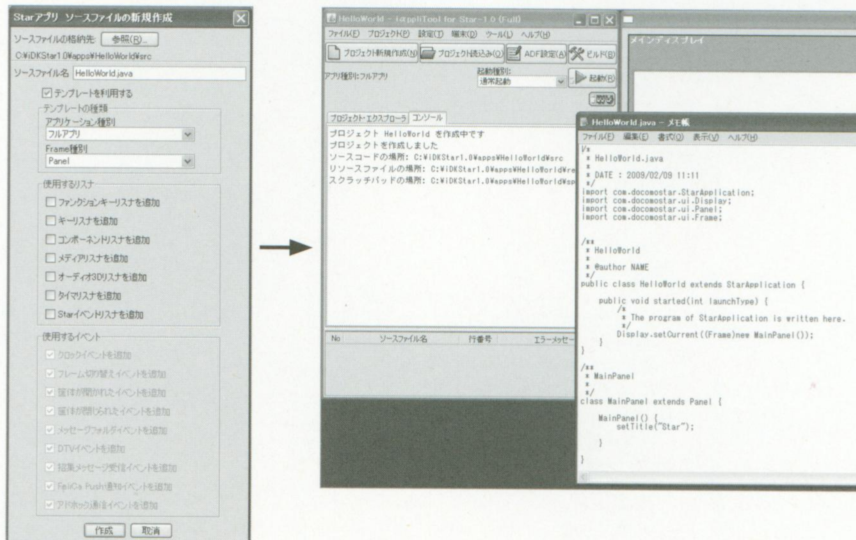


すると新規にプロジェクト「**HelloWorld**」が作成され、テキストエディタが開かれ「`HelloWorld.java`」内にコードを記述できるようになります。



Column» テンプレートの利用

「テンプレートを利用する」のチェックを入れて［作成］ボタンをクリックすると、ある程度のコードが自動的に生成されます。必要な記述だけを残し、新たに書き加えるなどでコーディングを省力化することができます。



▶ソースコードの記述 （HelloWorld.java）

それではテキストエディタを使って、2つのソースコード**HelloWorld.java**と**HelloCanvas.java**を記述しましょう。まずはiアプリの本体となる「HelloWorldクラス（HelloWorld.java）」です。StarプロファイルとDoJaプロファイルでは記述するソースコードが異なります。

本書では以降、**Star** **DoJa** というマークで分類をしながら解説します。

Star HelloWorld.java

```
import com.docomostar.StarApplication;
import com.docomostar.ui.Display;

// HelloWorld( 本体 )
public class HelloWorld extends StarApplication {
```

```
    // アプリの開始
    public void started(int launchType) {
        Display.setCurrent(new HelloCanvas());
    }
}
```

DoJa HelloWorld.java

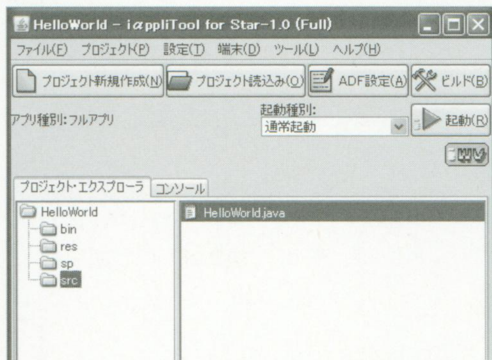
```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;

// HelloWorld(本体)
public class HelloWorld extends IApplication {

    // アプリの開始
    public void start() {
        Display.setCurrent(new HelloCanvas());
    }
}
```


英字や記号は全て半角文字を使用します。単語と単語の間に使う空白も全角文字を使ってはいけません。また、Javaのプログラムは大文字と小文字を区別しているため、「StarApplication」を「starapplication」と記述してはいけません。カッコの{}と()も正確に入力してください。

記述が終わったら、エディタでファイルを保存し閉じてください。iαppliToolの［プロジェクト・エクスプローラ］タブの「src」フォルダにファイル「HelloWorld.java」が保存されているかどうかを確認してください。



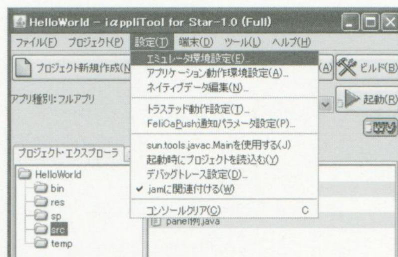
Column» テキストエディタの設定

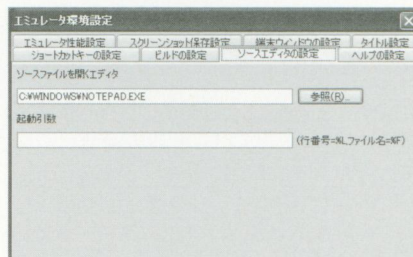
iαppliToolでは、通常はWindows標準のテキストエディタである**メモ帳**が利用できるようになっています。しかし「メモ帳」はコーディングに適しているとは言いがたいので、自分にあったテキストエディタを活用しましょう。フリーのテキストエディタのお勧めとしては、**TeraPad**や**サクラエディタ**があります。シンプルで使いやすく、ソースコード作成に便利な機能も備えています。

- **TeraPad**
  http://www5f.biglobe.ne.jp/~t-susumu/library/tpad.html

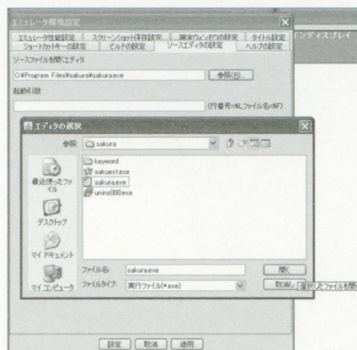
- **サクラエディタ**
  http://sakura_editor.at.infoseek.co.jp/

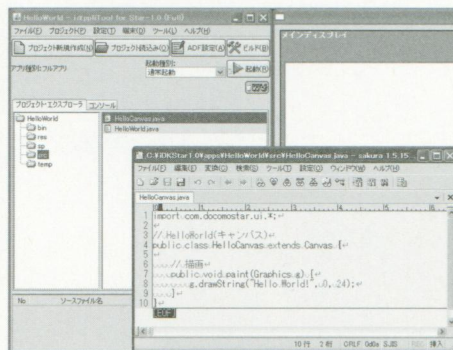
iαppliToolで利用するエディタの種類を変更するにはメニュー［設定］－［エミュレータ環境設定...］の［ソースエディタの設定］より選択することができます。［参照...］ボタンより対象とするテキストエディタを選んでください。





下記の画面は、「サクラエディタ」を関連付け、編集しているところです。

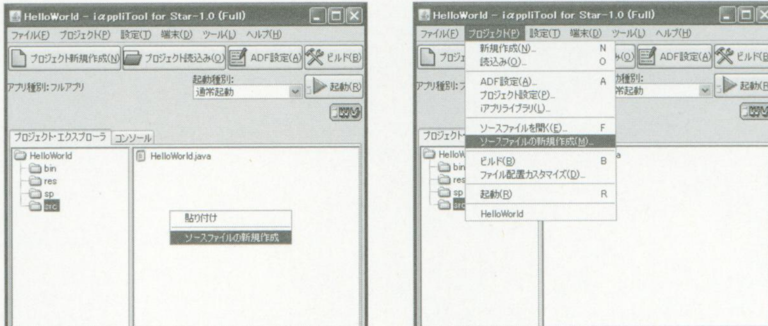




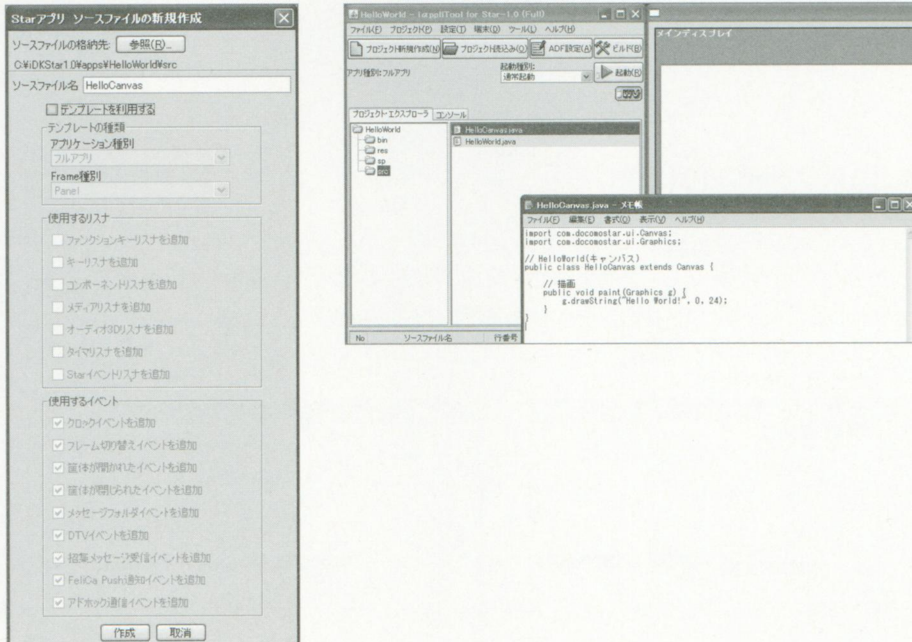
chapter
1
2
3
4
5
6
7
8
a

▶ソースコードの記述 （HelloCanvas.java）

次に2つ目の「HelloCanvasクラス（HelloCanvas.java）」を記述します。iαppliToolの［プロジェクト・エクスプローラ］タブの「src」フォルダ上で右クリックし、メニューより［ソースファイルの新規作成］を選ぶか、メニューより［プロジェクト］－［ソースファイルの新規作成...］を選びます。



次にソースファイルの新規作成ダイアログが開くので、ソースファイル名を「HelloCanvas.java」とし［作成］ボタンをクリックすると、エディタでHelloCanvas.javaを記述できるようになります。



HelloCanvas.javaは以下のように記述を行ってください。

Star HelloCanvas.java

```
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Graphics;

// HelloWorld(キャンバス)
public class HelloCanvas extends Canvas {

    // 描画
    public void paint(Graphics g) {
        g.drawString("Hello, World!",0,24);
    }
}
```

DoJa HelloCanvas.java

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;

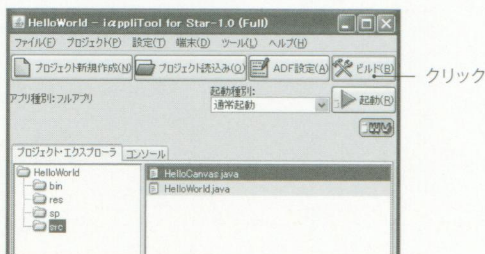
// HelloWorld(キャンバス)
public class HelloCanvas extends Canvas {

    // 描画
    public void paint(Graphics g) {
        g.drawString("Hello, World!",0,12);
    }
}
```

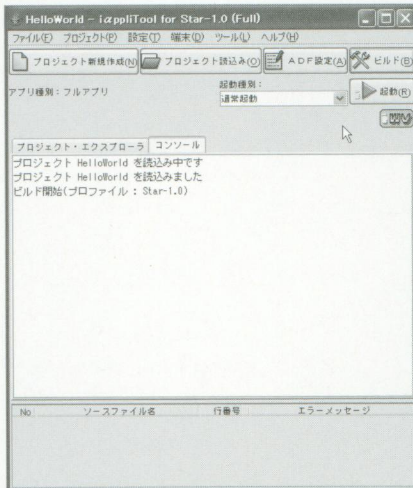
▶JARファイルの作成

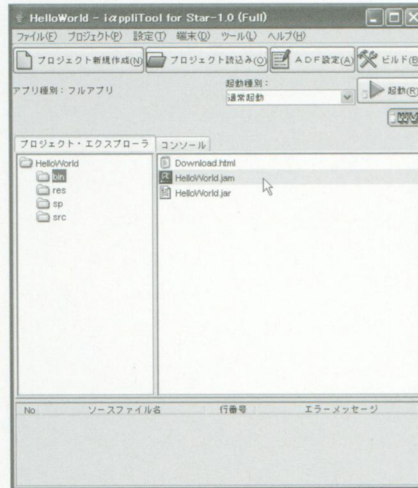
ソースコードが完成したら、iアプリの実行ファイルである**JARファイル**を作成します。iαppliToolの［プロジェクト・エクスプローラ］タブの「src」フォルダに「HelloWorld.java」「HelloCanvas.java」の2つのソースコードがあることを確認してから、メニューより［プロジェクト］－［ビルド］を選択するか、もしくは［ビルド］ボタンをクリックしてください。



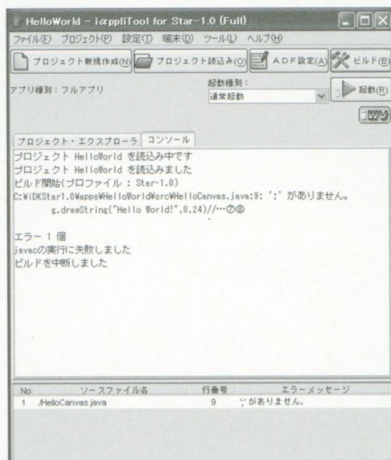


メッセージ領域に「ビルド終了」と表示されれば、成功です。Helloプロジェクトの「bin」フォルダ内に**iアプリ本体となるJARファイル「HelloWorld.jar」**が生成されています。





Column» エラーが表示された場合は

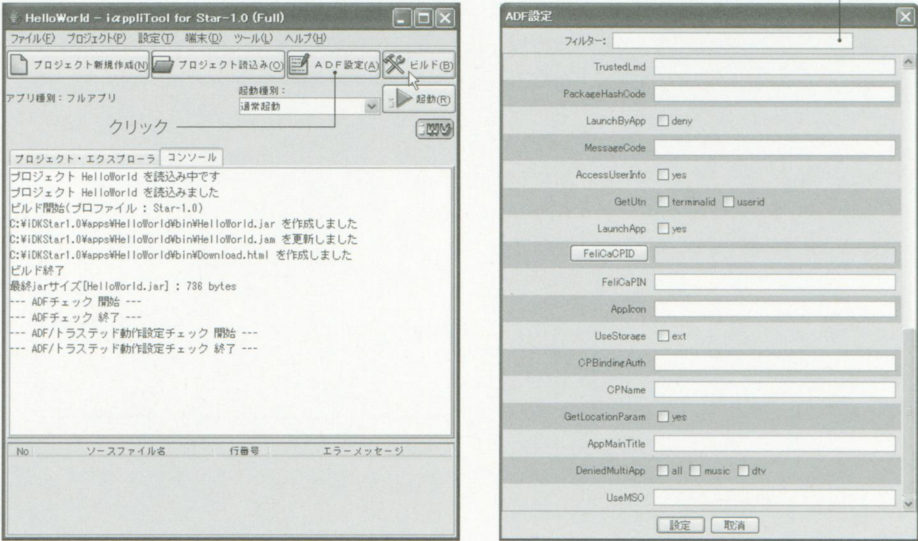


ビルド時に**エラー**と表示された時はソースコードの記述に誤っている部分があるので、テキストエディタで修正してビルドし直してください。エラーメッセージを読むと、何行目が失敗しているかが分かります。左のエラーメッセージは「HelloCanvas.javaの9行目で";"が抜けている」という意味です。

またJDKとして1.5以降を利用している時は、メニュー［設定］－［sun.tools.javac.Mainを使用する］にチェックを入れないとエラーとなる場合があります。

▶ADFの作成

JARファイルが完成したら、iアプリの属性ファイルである**ADF**を作成します。iαppliToolの[ADF設定]ボタンをクリックすると、「ADF設定」ダイアログが開きます。



入力必須項目は以下の表のとおりです。今回のアプリに必要な設定は、ビルド時に自動的に入力されています。入力後［設定］ボタンをクリックすることで、**HelloWorld.jam**に設定が反映されます。

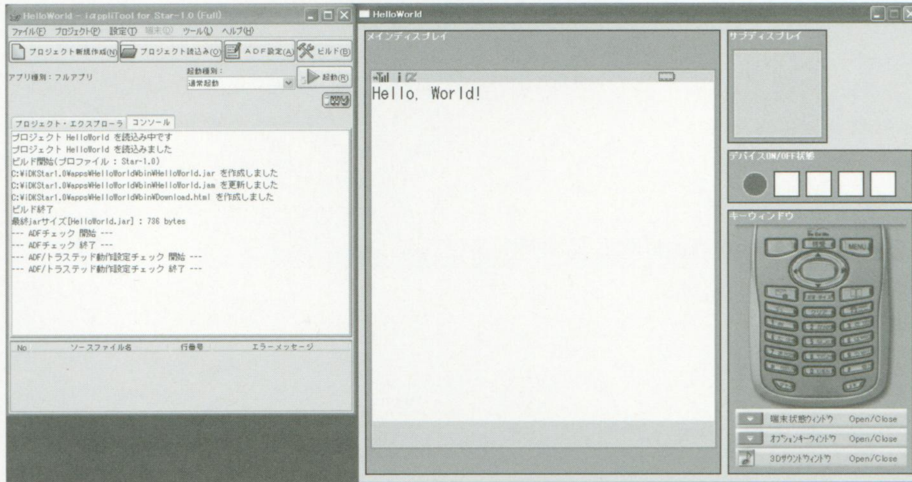
項目	説明
AppType	アプリケーション種別（Starプロファイル専用）
AppName	iアプリの名前
PackageURL	JARファイルの名前
AppSize	JARファイルの容量（バイト単位）
AppClass	iアプリの本体となるクラスの名前
LastModified	iアプリの最終更新日

ADFは設定項目が多いため、ウィンドウ上部の「フィルター：」を利用すると便利です。設定したい項目の頭文字を入力していくと自動的にADFの項目が絞り込まれていきます（たとえば「app」と入力すると app～ の項目のみが表示されます）。

これでiアプリの作成準備が整いました。

▶エミュレータでの実行

エミュレータは、パソコン上でiアプリを実行して動作確認を行うためのツールで、iαppliToolを起動すると一緒に表示される、携帯電話を模したウィンドウがそれにあたります。起動種別として「通常起動」を選択しているのを確認してから、［起動］ボタンを押します。



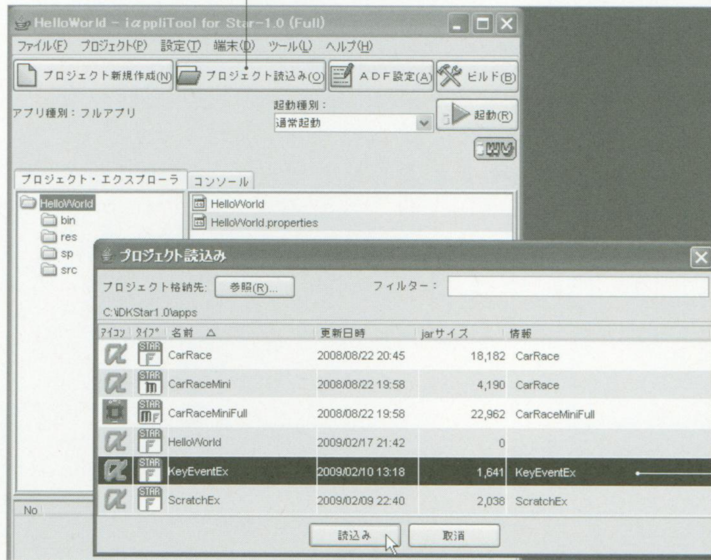
成功すれば、「メインディスプレイ」に「`Hello, World!`」と表示されることが確認できます。

▶プロジェクトフォルダについて

新しくプロジェクトを作成する、iαppli Development Kitがインストールされている「`iDKStar1.0`」または「`iDKDoJa5.1`」フォルダ内にある「`apps`」フォルダの中に、新しく「`HelloWorld`」フォルダが生成され、この中にはさらに次の4つのフォルダが生成されます。これはiαppliToolのプロジェクトエクスプローラでも閲覧できます。

フォルダ名	内容
binフォルダ	iアプリ（JARファイル）・ADF（JAMファイル）を生成
resフォルダ	イメージファイルやサウンドファイルを配置
spフォルダ	エミュレータのスクラッチパッドデータの保存
srcフォルダ	ソースコードを配置

作成済みのプロジェクトを開きたい時は、[プロジェクト読込み] ボタンをクリックしてください。プロジェクトの一覧が表示されるので、プロジェクト名を選択し [読込み] ボタンをクリックすることで、過去に作成したプロジェクトを呼び出すことができます。



2-4 iアプリ対応端末での実行

エミュレータで無事起動できることを確認したら、次は実際のiモード端末で実行してみましょう。

▶ダウンロード用HTMLの作成

iアプリをネットに公開してダウンロードできるようにするには、JARファイルとADFの他に、**ダウンロード用HTML**が必要です。iαpplitoolで作成したプロジェクトの「bin」フォルダには**Download.html**という名前でシンプルなダウンロード用HTMLが生成されています。

Download.html

```
<HTML>
<HEAD>
<TITLE>Download Page</TITLE>
</HEAD>
<BODY>
<OBJECT declare id="HelloWorld" …❶
        data="HelloWorld.jam"
        type="application/x-jam">
</OBJECT>
<BR>
<A ijam="#HelloWorld" href="notapplicable.html">DOWNLOAD</A> …❷
</BODY>
</HTML>
```

Download.html …❶：OBJECT要素

OBJECT要素は、ダウンロードするiアプリのADFの指定に使います。OBJECT要素のid属性にはA要素と対応付けるための任意の名前を指定し、data属性にはADFの名前を指定します

Download.html …❷：A要素

A要素は、iアプリをダウンロードするためのリンクの指定に使います。A要素のijam属性には、OBJECT要素のid属性で指定した名前に「#」を付けて指定します。A要素で指定したリンクをクリックした時、対応付けたOBJECT要素で指定しているiアプリをダウンロードします。href属性には、非対応端末からアクセスした時に開くHTMLのURLを指定します。

Column»　iモード対応HTML

ダウンロード用HTMLをはじめとするiモードで閲覧するWebページは**iモード対応HTML**で記述します。iモード対応HTMLは、Webページを記述するための記述言語HTMLから、iモード閲覧に必要な要素だけを選び簡略化したものです。基本的な記述手法については通常のHTMLの作成手法と同じです。iモード対応HTMLで利用できる要素（タグ）については、NTTドコモの以下のサイトを参照してください。

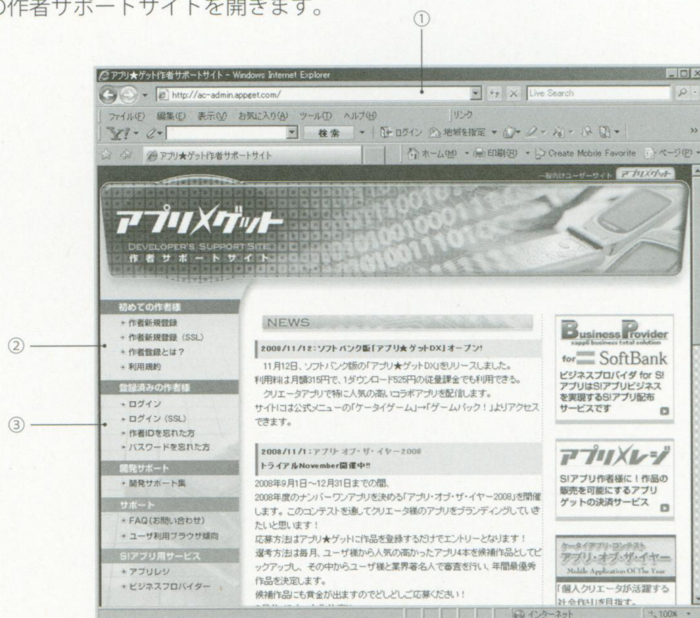
- iモード対応HTML
  http://www.nttdocomo.co.jp/service/imode/make/content/html/

▶iアプリ対応端末での実行

`HelloWorld`プロジェクトの「bin」フォルダに生成された「`HelloWorld.jar`」「`HelloWorld.jam`」「`Download.html`」の3つのファイルを、ネットにアップロードすることで、iアプリ対応端末で実行できるようになります。

アップロード方法はプロバイダごとに異なります。本書では「アプリ★ゲット」のホスティングサービスを利用してアプリをアップロードする方法について説明します。

① アプリ★ゲットの作者サポートサイトを開きます。



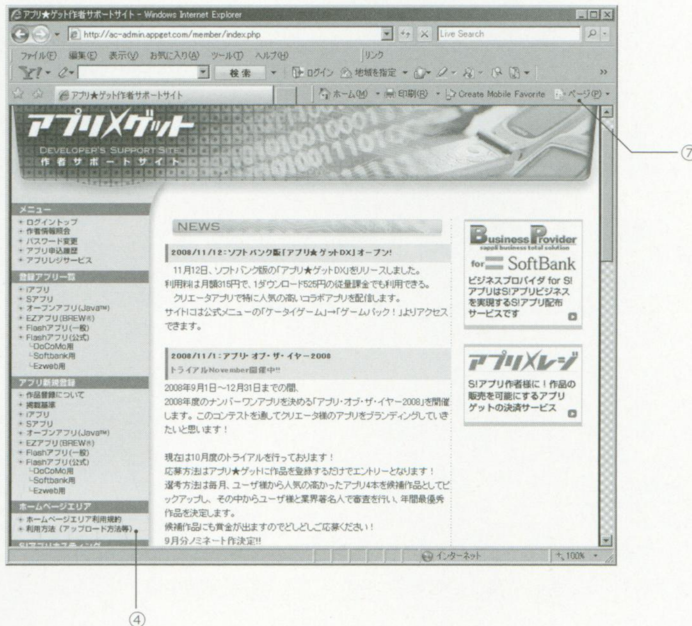
・アプリ★ゲット　作者サポートサイト
http://ac-admin.appget.com/

ページ左に各種ページメニューがあります。

② 作者登録していない人はページメニュー「作者新規登録」から作者登録を行ってください。

③ ページメニュー「ログイン」からログインします。

④ ページメニュー「ホームページエリア － 利用方法」をクリックします。



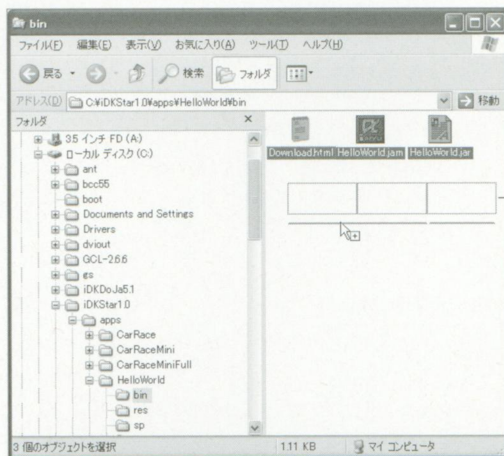
⑤ 「ホームページエリア － 利用方法」に記述されているURL「`ftp://ac-ftp.appget.com/public_html/`」にInternet Explorer 7（IE7）からアクセスします。

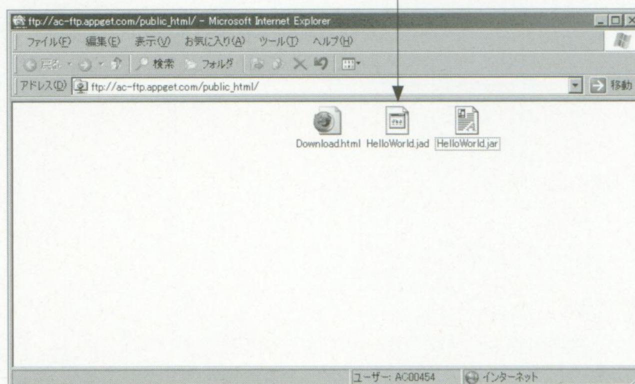
⑥ ユーザ名とパスワードを求められるので、作者登録した時の作者IDとパスワードを入力します。

⑦ 「ページをクリックして、エクスプローラでFTPサイトを開くをクリックしてください。」と表示されるので、IE7のタブの左にある「ページ」をクリックし、「エクスプローラでFTPサイトを開く」を選択します。

⑧ FTP接続してエクスプローラで開いた場所に、ドラッグ&ドロップでファイルをアップロードします。

以上でアップロードは終了です。





アップロードしたファイルにアクセスするためのURLは「http://ac.appget.com/作者ID/ファイル名」です。実機でダウンロード用HTMLファイルにアクセスし、ダウンロード用のリンクをクリックすることで、iアプリをダウンロードして実行することができます。

Column» アプリの作品登録

「アプリ★ゲット」のホスティングサービスにiアプリをアップロードしても、アプリ★ゲットのiアプリ紹介ページには載りません。ダウンロードHTMLのURLを知っている人だけが利用できます。

アプリ紹介ページに載せてもらいたい時は、作者サポートサイトでメニュー「アプリ新規登録」-「iアプリ」で作品登録を行ってください。

3

基本API

chapter
1
2
3
4
5
6
7
8
a

本章では、まず2章で作成した「Hello, World!」を表示するだけのiアプリのコードについて詳しく解説します。このiアプリの作成を通して、開発の基礎を習得してください。

3.2節以降では、プログラミングの基本となる文字列やイメージの描画と、キーイベントの処理およびサウンドの再生について解説します。

3-1 iアプリの基本（Hello, World!）



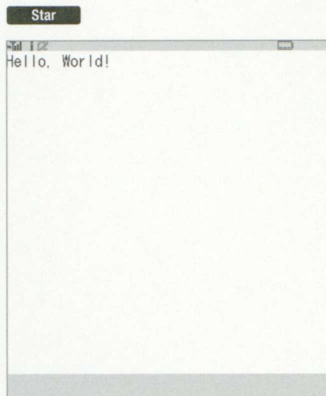
Star
http://book.mycom.co.jp/support/pc/star/31s.html



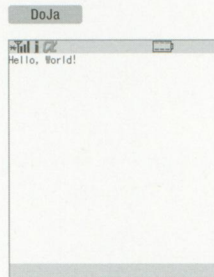
DoJa
http://book.mycom.co.jp/support/pc/star/d/31d.html

Chapter 2で作成した「Hello, World!」プログラムのコーディング内容について解説をします。このiアプリを起動すると「Hello, World!」とのみ画面に表示されます。このiアプリの作成方法を通して、iアプリの作成方法をマスターしてください。

Star



DoJa



▶HelloWorldクラス

HelloWorldクラスは、**プログラムの本体**となるクラスです。

Star HelloWorld.java

```
import com.docomostar.StarApplication; // …❶
import com.docomostar.ui.Display;      //

// HelloWorld( 本体 )
public class HelloWorld extends StarApplication { // …❷，❸

    // アプリの開始
    public void started(int launchType) {      // …❹
        Display.setCurrent(new HelloCanvas()); // …❺
    }
}
```

DoJa HelloWorld.java

```
import com.nttdocomo.ui.IApplication;       // …❶
import com.nttdocomo.ui.Display;            //

// HelloWorld(本体)
public class HelloWorld extends IApplication { // …❷,❸

    // アプリの開始
    public void start() {                      // …❹
        Display.setCurrent(new HelloCanvas()); // …❺
    }
}
```

HelloWorld …❶：パッケージ宣言

importは、**パッケージ**を使う時に使う予約語で、Starプロファイル用のHelloWorldの場合は「import com.docomostarパッケージ」「import com.docomostar.uiパッケージ」の持つクラスを使うことを宣言しています。StarApplicationクラス、Displayクラスがこれらパッケージに含まれています。

HelloWorld …❷：アクセス修飾子

publicは**アクセス修飾子**と呼ばれるもので、他のクラスや他のパッケージからのアクセスを許可するかどうかを指定するもので、private、**指定なし**、protected、publicの4種類があります。

privateは同じクラスからのアクセスのみ許可します。**指定なし**は同じパッケージ内のクラスからのみアクセスを許可します。protectedは同じクラスかサブクラスからのアクセスのみ許可します。publicは全てのアクセスを許可します。

	private	指定なし	protected	public
同じクラス	○	○	○	○
同じパッケージ内のサブクラス	×	○	○	○
同じパッケージ内の非サブクラス	×	○	×	○
他のパッケージ内のサブクラス	×	×	○	○
他のパッケージ内の非サブクラス	×	×	×	○

※サブクラスとは継承先のクラスのことです。

Column» ステートメント

プログラミング言語では、1つの命令のことを**ステートメント**と呼びます。Javaのステートメントの終わりには必ず「;」を指定します。

HelloWorld ...❸：クラス定義

Javaではプログラムの最小単位を**クラス**と呼びます。クラスを指定しているのがclassというキーワードで、それに続く「HelloWorld」がクラス名となります。

クラス定義の書式は次の通りです。

［クラス定義の書式］

```
アクセス修飾子 class クラス名 extends 継承元クラス名 {
    クラスの中身
}
```

Starプロファイルでは「StarApplicationクラス」、DoJaプロファイルでは「IApplicationクラス」がiアプリの土台となるクラスで、全てのiアプリはこれらクラスを**継承**しています。

継承とは既存のクラスを元に新しいクラスを作成するしくみで、新しいクラスは既存のクラスが持っている機能を受け継ぎ、さらに独自機能を追加することができます。「extends クラス名」がクラスを継承することの宣言で、その後ろの{から最後の}までがクラスの中身となります。

HelloWorld ...❹：メソッド定義

クラスの中身の部分には、クラスで行う処理を記述します。Javaではクラスの持つ処理のことを**メソッド**と呼びます。クラスに何か行わせたい時は、対応するメソッドを呼び出します。

単純なiアプリのプログラムの場合、Starプロファイルではstarted()、DoJaプロファイルではstart()という名前のメソッドが1つだけ存在します。iアプリを起動した時、まず最初にこのメソッドの中に記述されている処理を実行します。

Star ［StarApplicationクラス］

void started(*launchType*)

［解説］ アプリケーション起動時に呼ばれる

［引数］ *launchType* 起動元種別：int型

DoJa ［IApplicationクラス］

void start()

［解説］ アプリケーション起動時に呼ばれる

引数*launchType*は起動元種別（メニューやメーラなど）に応じて処理を変更したい時に使います。

メソッド定義の書式は次の通りです。

```
アクセス修飾子 戻り値の型 メソッド名 ( 引数の型１ 引数の名前１, …) {
      処理
      return 戻り値
}
```

メソッドは処理を実行した結果として**戻り値**を返します。どの値を戻り値として返すかは、メソッド内のreturn文で記述します。戻り値がない時はvoidと記述します。

HelloWorld ...❺：Displayクラス

Displayクラスは実画面に表示する画面を指定するクラスです。今回はHelloCanvasクラスのオブジェクトを指定しています。

［Displayクラス］

static void setCurrent(*frame*)

［解説］ 実画面に表示する画面を指定

［引数］ *frame* 　実画面に表示する画面：Frame型

Column» インスタンスとスタティック

「static」が付いているものは、オブジェクトでなくクラスで保持するフィールド変数やメソッドで、全オブジェクト共通で同じデータ、同じ操作を使いたい時に利用します。

staticなしの方を**インスタンスフィールド**、**インスタンスメソッド**、staticありの方を**スタティックフィールド**、**スタティックメソッド**と呼びます。

▶HelloCanvasクラス

HelloCanvasクラスは、キャンバスとなるクラスです。

Star HelloCanvas.java

```
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Graphics;

// HelloWorld(キャンバス)
public class HelloCanvas extends Canvas {  // …❶

    // 描画
    public void paint(Graphics g) {
        g.drawString("Hello, World!",0,24); // …❷,❸
    }
}
```

DoJa HelloCanvas.java

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;

// HelloWorld(キャンバス)
public class HelloCanvas extends Canvas {  // …❶

    // 描画
    public void paint(Graphics g) {
        g.drawString("Hello, World!",0,12); // …❷,❸
    }
}
```

HelloCanvas …❶：Canvasクラス

Canvasクラスはキャンバスの土台となる機能を持っているクラスで、キャンバスとなるクラスはこのクラスを継承します。Canvasを継承したクラスは、必ずpaint()メソッドを持つ（オーバーライドする）必要があります。

paint()メソッドは、iアプリ起動時や再描画が必要な時に呼ばれます。このメソッドに引数として渡されるGraphicsクラスのオブジェクトを操作することにより、画面に文字列や絵などを描画できます。

［Canvasクラス］

void paint(*g*)

［解説］ 描画時に呼ばれる
［引数］ *g*　グラフィックス：Graphics型

HelloCanvas ...❷：Graphicsクラス

Graphicsクラスは、画面に文字列や絵などを表示するためのクラスです。文字列を表示するにはGraphicsクラスのdrawString()メソッドを使います。

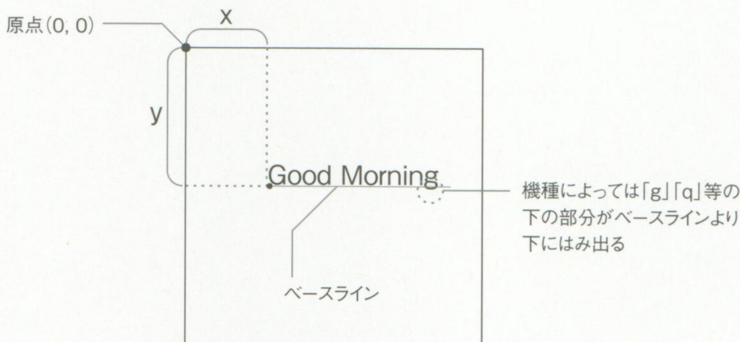
［Graphicsクラス］

void drawString(*text, x, y*)

［解説］ 文字列を描画

［引数］ *text*　テキスト：String型
x　X座標：　int型
y　Y座標：　int型

引数*x*（X座標）には画面左から何ドット右に移動した位置か、*y*（Y座標）には画面上から何ドット下に移動した位置かを示します。指定したXY座標は、文字列の左上でなく左下（ベースラインの左隅）の座標となります。



HelloCanvas ...❸：文字列

drawString()メソッドに渡している「"Hello,World!"」は文字列です。Java言語では「"」で囲んだ文字列がString型の文字列オブジェクトとなります。文字列は + 演算子を使うことで、左右の文字列を結合することができます。さらに、文字列と数値に対して + を使うと、自動的に数値を文字列に変換して結合することもできます。

Column»　コメント

コメントとはプログラムの注意書きのことです。ソースコードを読みやすくするために使います。HelloWorld.javaでは「//HelloWorld」などがコメントにあたります。

- 1行のコメント　　…「//」から行末まで
- 複数行のコメント …「/*」から「*/」まで

3-2 文字列の描画



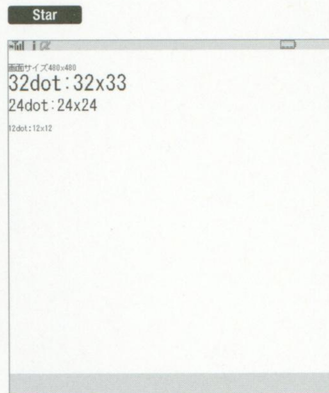
Star
http://book.mycom.co.jp/support/pc/star/32s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/32d.html

文字列を表示するプログラムを作ります。**画面サイズ**（幅x高）と**任意のフォントの全角1文字のサイズ**（幅x高さ）を表示します。

Star



DoJa



今回のプログラムは、次の2つのクラスで構成されています。

- **StringEx**クラス（**StringEx.java**）
- **StringCanvas**クラス（**StringCanvas.java**）

プロジェクト名「**StringEx**」でプロジェクトを作成してください。

▶StringExクラス

StringExクラスは、プログラムの本体となるクラスです。

Star StringEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// 文字の描画 ( 本体 )
```

Next page.

```
public class StringEx extends StarApplication {  // ⓑ

    // アプリの開始
    public void started(int launchType) { // ⓒ
        Display.setCurrent(new StringCanvas());
    }
}
```

DoJa StringEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class StringEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶StringCanvasクラス

StringCanvasクラスは、キャンバスとなるクラスです。

Star StringCanvas.java

```
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Font;
import com.docomostar.ui.Graphics;

// 文字の描画 ( キャンバス )
public class StringCanvas extends Canvas {

    // 描画
    public void paint(Graphics g) {
        Font font;

        // 画面の塗り潰し
        g.setColor(g.getColorOfName(Graphics.WHITE)); // …❶
        g.fillRect(0,0,getWidth(),getHeight());       // …❷



        // 画面サイズ
        g.setColor(g.getColorOfName(Graphics.BLACK));
```

```
        font = Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,12);
        g.setFont(font);
        g.drawString("画面サイズ"+getWidth()+"x"+getHeight(),0,30*1);

        // 32dot フォントサイズ
        font = Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,32); // …❸
        g.setFont(font);                                           // …❹
        g.drawString("32dot:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*2);                              // …❺

        // 24dot フォントサイズ
        font = Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,24);
        g.setFont(font);
        g.drawString("24dot:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*3);

        // 12dot フォントサイズ
        font = Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,12);
        g.setFont(font);
        g.drawString("12dot:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*4);
    }
}
```

DoJa StringCanvas.java

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;

// 文字の描画(キャンバス)
public class StringCanvas extends Canvas {

    // 描画
    public void paint(Graphics g) {
        Font font;

        // 画面の塗り潰し
        g.setColor(g.getColorOfName(Graphics.WHITE)); // …❶
        g.fillRect(0,0,getWidth(),getHeight());       // …❷

        // 画面サイズ
        g.setColor(g.getColorOfName(Graphics.BLACK));
        g.drawString("画面サイズ"+getWidth()+"x"+getHeight(),0,30*1);

        // LARGE フォントサイズ
        font = Font.getFont(Font.SIZE_LARGE);         // …❸
```

Next page. ⬇

```
        g.setFont(font);                          // ...❹
        g.drawString("LARGE:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*2);             // ...❺

        // MEDIUMフォントサイズ
        font = Font.getFont(Font.SIZE_MEDIUM);
        g.setFont(font);
        g.drawString("MEDIUM:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*3);

        // SMALLフォントサイズ
        font = Font.getFont(Font.SIZE_SMALL);
        g.setFont(font);
        g.drawString("SMALL:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*4);

        // TINYフォントサイズ
        font = Font.getFont(Font.SIZE_TINY);
        g.setFont(font);
        g.drawString("TINY:"+font.stringWidth("あ")+"x"+
            font.getHeight(),0,30*5);
    }
}
```

StringCanvas ...❶：色の指定

文字列や図形の色を指定するには、GraphicsクラスのsetColor()メソッドを使います。このメソッドを使用すると、それ以降の文字列や図形を描画する時に指定された色が適用されます。

［Graphicsクラス］

void setColor(*color*)

［解説］ 色の指定

［引数］ *color* 色値：int型

引数の*color*（色値）は、GraphicsクラスのgetColorOfRGB()メソッド、またはgetColorOfName()メソッドで取得します。

［Graphicsクラス］

static int getColorOfRGB(*red, green, blue*)

［解説］ 色値の取得

［引数］ *red* 赤（0〜255）：int型

green 緑（0〜255）：int型

blue 青（0〜255）：int型

［戻り値］ 色値

［Graphicsクラス］

static int getColorOfName(*color*)

［解説］ 色値の取得

［引数］ *color* 色定数：int型

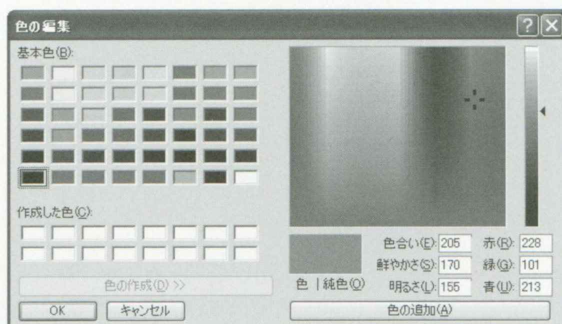
［戻り値］ 色値

引数の*color*（色定数）には、次の定数を指定します。

Graphics.AQUA	水色
Graphics.BLACK	黒
Graphics.BLUE	青
Graphics.FUCHSIA	紫色
Graphics.GRAY	灰色
Graphics.GREEN	暗い緑色
Graphics.LIME	緑色
Graphics.MAROON	暗い赤色
Graphics.NAVY	暗い青色
Graphics.OLIVE	暗い黄色
Graphics.PURPLE	暗い紫色
Graphics.RED	赤色
Graphics.SILVER	銀色
Graphics.TEAL	暗い水色
Graphics.WHITE	白色
Graphics.YELLOW	黄色

Column» RGB値の調べ方

自分の使いたい色のRGB値を調べるには、画像処理関連ソフトのカラーパレットを使うとよいでしょう。Windowsの「ペイント」なら、メニューで［色］－［色の編集...］を選ぶと「色の編集」ダイアログが開くので、そこで［色の追加］ボタンをクリックすると色の選択画面が表示されます。あとはマウスで色を選べば、テキストフィールドに赤（R）と緑（G）と青（B）のRGB数値が表示されます。



StringCanvas ...❷：画面の塗り潰し

四角形を描画し、指定した色で塗り潰すには、GraphicsクラスのfillRect()メソッドを使います。

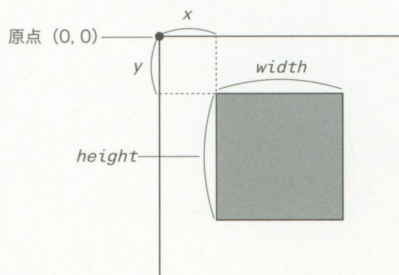
［Graphicsクラス］

void fillRect(*x, y, width, height***)**

［解説］ 四角形の描画

［引数］	*x*	X座標：int型	*width*	幅　：int型
	y	Y座標：int型	*height*	高さ：int型

引数の*x*（X座標）には画面左端から何ドット右に移動した位置、*y*（Y座標）には画面上端から何ドット下に移動した位置を指定します。



画面の幅と高さを取得するには、CanvasクラスのgetWidth()メソッドとgetHeight()メソッドを使います。

［Canvasクラス］

int getWidth()

［解説］ キャンバスの幅の取得
［戻り値］ キャンバスの幅

［Canvasクラス］

int getHeight()

［解説］ キャンバスの高さの取得
［戻り値］ キャンバスの高さ

StringCanvas ...❸：フォントの生成

Star フォントを生成するには、FontクラスのgetFont()メソッドを使います。

Star ［Fontクラス］

static Font getFont(*type*, *size*)

［解説］ フォントの取得
［引数］ *type* フォントタイプ：int型
size フォントサイズ：int型
［戻り値］ フォントオブジェクト

引数の*type*（フォントタイプ）には次の定数を論理和で指定します。

フォントフェイス	Font.FACE_MONOSPACE	モノスペースフォント（オプション）
	Font.FACE_PROPORTIONAL	プロポーショナルフォント（オプション）
	Font.FACE_SYSTEM	システムフォント
フォントスタイル	Font.STYLE_BOLD	ボールド（オプション）
	Font.STYLE_BOLDITALIC	ボールドイタリックスタイル（オプション）
	Font.STYLE_ITALIC	イタリック（オプション）
	Font.STYLE_PLAIN	プレーン

DoJa フォントを生成するには、FontクラスのgetFont()メソッドで取得します。

DoJa ［Fontクラス］

static Font getFont(*type*)

［解説］ フォントの取得

［引数］ *type* フォントタイプ：int型

［戻り値］ フォントオブジェクト

引数の*type*（フォントタイプ）には次の定数を論理和で指定します。

フォントサイズ	Font.SIZE_LARGE	フォントのラージサイズ
	Font.SIZE_MEDIUM	ミディアムサイズ
	Font.SIZE_SMALL	スモールサイズ
	Font.SIZE_TINY	タイニーサイズ
フォントフェイス	Font.FACE_MONOSPACE	モノスペースフォント（オプション）
	Font.FACE_PROPORTIONAL	プロポーショナルフォント（オプション）
	Font.FACE_SYSTEM	システムフォント
フォントスタイル	Font.STYLE_BOLD	ボールド（オプション）
	Font.STYLE_BOLDITALIC	ボールドイタリックスタイル（オプション）
	Font.STYLE_ITALIC	イタリック（オプション）
	Font.STYLE_PLAIN	プレーン

指定したフォントサイズが具体的に何ドットになるかは機種依存です。詳しくは次のサイトを参照してください。Star-1.0/DoJa-5.1プロファイル対応端末については本書の付録a-1（346ページ）に一覧を掲載しましたので、あわせて確認してください。

・iアプリ　端末スペック一覧
http://www.nttdocomo.co.jp/service/imode/make/content/spec/iappli/

StringCanvas ...❹：フォントの指定

フォントを指定するには、GraphicsクラスのsetFont()メソッドを使います。

［Graphicsクラス］

void setFont(*font*)

［解説］ フォントの指定
［引数］ *font* フォントオブジェクト：Font型

StringCanvas ...❺：文字列の幅と高さの取得

文字列の幅を調べるにはFontクラスのstringWidth()メソッド、高さを調べるにはgetHeight()メソッドを使います。

［Fontクラス］

int stringWidth(*str*)

［解説］ 文字列の幅の取得
［引数］ *str* 文字列：String型
［戻り値］ 幅

［Fontクラス］

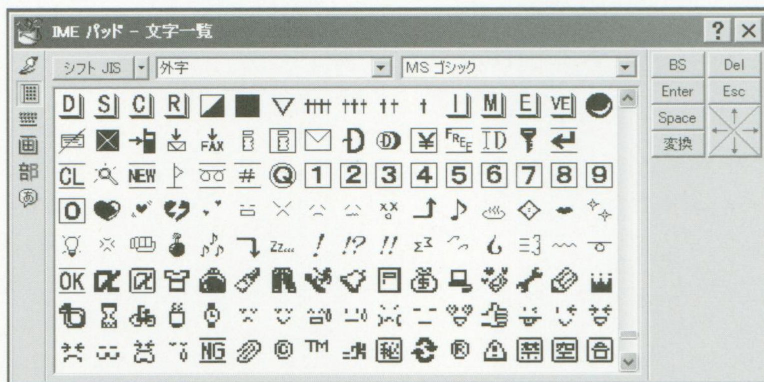
int getHeight()

［解説］ 文字列の高さの取得
［戻り値］ 高さ

Column» 絵文字

iモード対応絵文字フォントはiαppli Development Kitのインストール時にパソコンにインストールされます。

パソコンでこの絵文字を使いたい時は、テキストエディタで「がいじ」という文字列を打った後、改行を押さないで、部首・コード変換（IMEキー設定では［F5］、ATOKキー設定では［Shift］+［F6］）を押します。外字入力用のダイアログが開くので、絵文字を選択してください。



絵文字は通常の文字列と同様に、drawString()メソッドで表示することができます。

```
g.drawString("♠♥♣♦",0,180);
```

初期状態では絵文字は、スペードなら黒、ハートなら赤、のようにデフォルト色で描画されます。setColor()メソッドで指定した色を反映させたい時は、setPictoColorEnabled()メソッドを使ってください。

［Graphicsクラス］

void setPictoColorEnabled(*flag*)

［解説］指定色で描画するかどうかを指定

［引数］*flag*　　指定色で描画するかどうか：boolean型

・iモード対応絵文字
http://www.nttdocomo.co.jp/service/imode/make/content/pictograph/

3-3 図形の描画

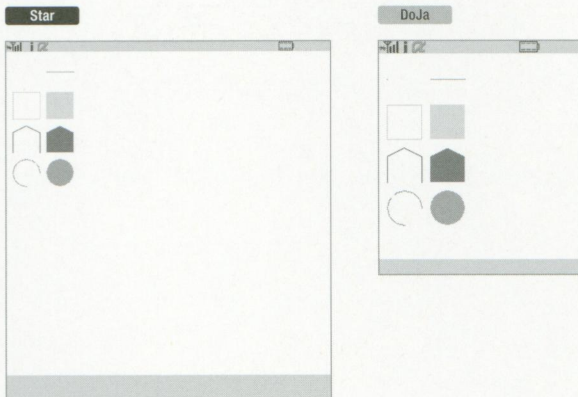


Star
http://book.mycom.co.jp/support/pc/star/33s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/33d.html

ピクセル、ライン、四角形、ポリライン、ポリゴン、円弧を描画するプログラムを作ります。



今回のプログラムは、次の2つのクラスで構成されています。

- **GraphicsEx**クラス（**GraphicsEx.java**）
- **GraphicsCanvas**クラス（**GraphicsCanvas.java**）

プロジェクト名「**GraphicsEx**」でプロジェクトを作成してください。

▶GraphicsExクラス

GraphicsExクラスは、プログラムの本体となるクラスです。

Star GraphicsEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //


// 図形の描画（本体）
```

Next page.

```
public class GraphicsEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {          // ⓒ
        Display.setCurrent(new GraphicsCanvas());
    }
}
```

DoJa GraphicsEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class GraphicsEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶GraphicsCanvasクラス

GraphicsCanvasクラスは、キャンバスとなるクラスです。

Star GraphicsCanvas.java

```
import com.docomostar.ui.Canvas;   // ⓐ
import com.docomostar.ui.Graphics; //

// 図形の描画（キャンバス）
public class GraphicsCanvas extends Canvas {

    // 描画
    public void paint(Graphics g) {
        // ピクセルの描画 ...❶
        g.setColor(g.getColorOfRGB(255,0,0));
        g.setPixel(10,30);

        // ラインの描画 ...❷
        g.setColor(g.getColorOfRGB(255,100,100));
        g.drawLine(60,30,100,30);

        // 四角形の描画 ...❸
```

```
            g.setColor(g.getColorOfRGB(0,255,0));
            g.drawRect(10,60,40,40);

            // 四角形の塗り潰し ...❹
            g.setColor(g.getColorOfRGB(100,255,100));
            g.fillRect(60,60,40,40);

            // ポリラインの描画 ...❺
            g.setColor(g.getColorOfRGB(0,0,255));
            int[] dx0 = { 10, 10, 30, 50, 50};
            int[] dy0 = {150,120,110,120,150};
            g.drawPolyline(dx0,dy0,dx0.length);

            // ポリゴンの描画 ...❻
            g.setColor(g.getColorOfRGB(100,100,255));
            int[] dx1 = { 60, 60, 80,100,100};
            int[] dy1 = {150,120,110,120,150};
            g.fillPolygon(dx1,dy1,5);

            // 円弧の描画 ...❼
            g.setColor(g.getColorOfRGB(255,0,255));
            g.drawArc(10,160,40,40,0,270);

            // 円弧の塗り潰し ...❽
            g.setColor(g.getColorOfRGB(255,100,255));
            g.fillArc(60,160,40,40,0,360);
        }
    }
```

DoJa GraphicsCanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;
```

GraphicsCanvas ...❶：ピクセルの描画

ピクセルを描画する時は、GraphicsクラスのsetPixel()メソッドを使います。

［Graphicsクラス］

void setPixel(*x*, *y*)

［解説］ ピクセルの描画

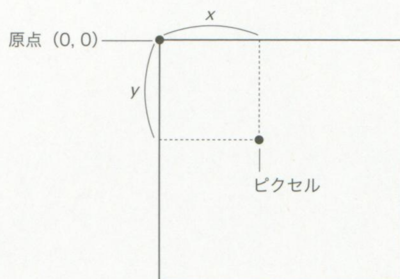
［引数］ *x* X座標：int型

y Y座標：int型

［Graphicsクラス］

void setPixel(*x, y, color*)

［解説］	ピクセルの描画	
［引数］	*x*	X座標：int型
	y	Y座標：int型
	color	色値： int型



GraphicsCanvas …❷：ラインの描画

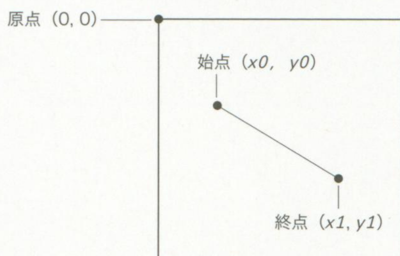
ラインを表示する時は、GraphicsクラスのdrawLine()メソッドを使います。

［Graphicsクラス］

void drawLine(*x0, y0, x1, y1*)

［解説］	ラインの描画	
［引数］	*x0*	始点のX座標：int型
	y0	始点のY座標：int型
	x1	終点のX座標：int型
	y1	終点のY座標：int型

表示される線は、始点（*x0*，*y0*）と終点（*x1*，*y1*）を結ぶ直線になります。



GraphicsCanvas ...❸：四角形の描画

四角形を描画する時は、GraphicsクラスのdrawRect()メソッドを使います。

［Graphicsクラス］

void drawRect(*x, y, width, height***)**

［解説］ 四角形の描画

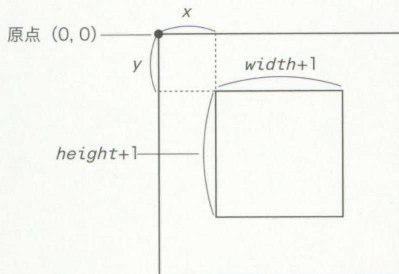
［引数］ *x* X座標：int型

y Y座標：int型

width 幅：int型

height 高さ：int型

左上（*x*, *y*）と右下（*x* + *width*, *y* + *height*）で指定される四角形（幅:*width* + 1、高さ:*height* + 1）を描画します。幅と高さは必ず0以上にしてください。



GraphicsCanvas ...❹：四角形の塗り潰し

四角形を塗り潰す時は、GraphicsクラスのfillRect()メソッドを使います。

［Graphicsクラス］

void fillRect(*x, y, width, height***)**

［解説］ 四角形の塗り潰し

x	X座標：int型	*width*	幅：int型
y	Y座標：int型	*height*	高さ：int型

左上（*x*, *y*）と右下（*x* + *width* - 1, *y* + *height* - 1）で指定される四角形（幅:*width*、高さ:*height*）を塗りつぶします。描画イメージについては066ページを参考にしてください。幅と高さは必ず0以上にしてください。

GraphicsCanvas …❺：ポリラインの描画

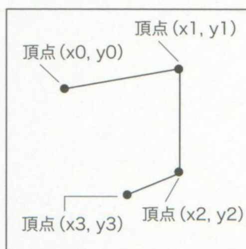
ポリライン（折れ線）を描画する時は、GraphicsクラスのdrawPolyline()メソッドを使います。

［Graphicsクラス］

void drawPolyline(*xPoints, yPoints, nPoints***)**

［解説］	ポリラインの描画		
［引数］	*xPoints*	X座標の配列：	int[]型
	yPoints	Y座標の配列：	int[]型
	nPoints	頂点数：	int型

引数の*xPoints*（X座標の配列）と*yPoints*（Y座標の配列）には、折れ線の頂点の座標を設定します。3つ目の引数*nPoints*には、頂点数を指定します。



GraphicsCanvas …❻：ポリゴンの描画

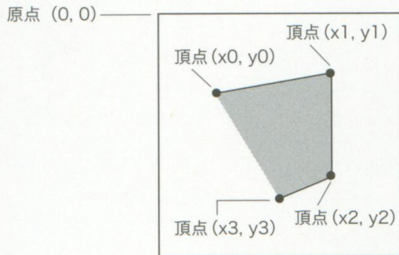
ポリゴン（多角形）を描画する時は、GraphicsクラスのfillPolygon()メソッドを使います。

［Graphicsクラス］

void fillPolygon(*xPoints, yPoints, nPoints***)**

［解説］	ポリゴンの描画		
［引数］	*xPoints*	X座標の配列：	int[]型
	yPoints	Y座標の配列：	int[]型
	nPoints	頂点数：	int型

引数の*xPoints*（X座標の配列）と*yPoints*（Y座標の配列）には、多角形の頂点の座標を設定します。3つ目の引数*nPoints*には、頂点数を指定します。



GraphicsCanvas ...❼：円弧の描画

円弧を描画する時は、GraphicsクラスのdrawArc()メソッドを使います。

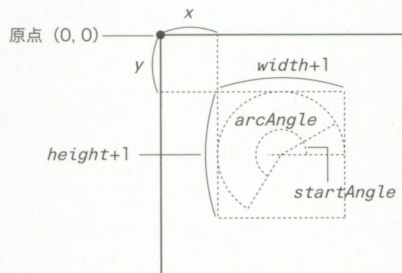
［Graphicsクラス］

void drawArc(*x, y, width, height, startAngle, arcAngle***)**

［解説］ 円弧の描画

［引数］	*x*	X座標：int型
	y	Y座標：int型
	width	幅：　int型
	height	高さ：　int型
	startAngle	弧の始点の角度（0〜360）：　int型
	arcAngle	弧の始点からの角度（0〜360）：int型

左上（*x*, *y*）と右下（*x* + *width*, *y* + *height*）で指定される領域（幅:*width* + 1、高さ:*height* + 1）内に円弧を描画します。弧の始点の角度と弧の始点からの角度は0〜360の値で指定します。



GraphicsCanvas ...❽：円弧の塗り潰し

円弧を塗り潰す時は、Graphicsクラスの`fillArc()`メソッドを使います。

［Graphicsクラス］

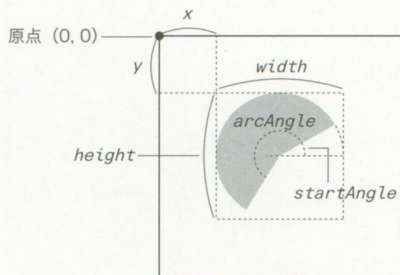
void fillArc(*x, y, width, height, startAngle, arcAngle*)

［解説］ 円弧の塗り潰し

［引数］

x	X座標：	int型
y	Y座標：	int型
width	幅：	int型
height	高さ：	int型
startAngle	弧の始点の角度（0〜360）：	int型
arcAngle	弧の始点からの角度（0〜360）：	int型

左上（*x*, *y*）と右下（*x* + *width* - 1, *y* + *height* - 1）で指定される領域（幅:*width*、高さ:*height*）内に円弧を塗り潰します。弧の始点の角度と弧の始点からの角度は0〜360の値で指定します。



Column» 変数の型

変数の型には、数値や`true/false`といった基本的な値を保持する**基本データ型**と、オブジェクトの参照を保持する**オブジェクト型**があります。基本データ型の変数には、変数の型に`char`、`byte`、`short`、`int`、`long`、`float`、`double`、`boolean`、変数の中身に「数値」「`true/false`」、オブジェクト型の変数では、変数の型に「クラス」、変数の中身に「オブジェクトの参照」を指定します。

	型	ビット長	説明	範囲	初期値
基本データ型	char	16	文字	Unicodeの1文字	' '（半角空白）
	byte	8	符号付き整数	-128〜127	0
	short	16	符号付き整数	-32,768〜32,767	0
	int	32	符号付き整数	-2,147,483,648〜2,147,483,648-1	0
	long	64	符号付き整数	-9.22337204*10^18〜9.22337204*10^18-1	0
	Float	32	浮動小数点	単精度浮動小数点	0.0
	Double	64	浮動小数点	倍精度浮動小数点	0.0
	Boolean	1	真偽値	trueまたはfalse	false
オブジェクト型	クラス	-	オブジェクトの参照	-	null

charは1つの文字を格納するデータ型です。文字はJavaの内部でUnicode（ユニコード）」と呼ばれる文字コードで格納されます。Unicodeは、世界中の文字を全て16ビットで表しているデータ形式です。代入する時には、文字をシングルクォーテーション「'」で囲みます。

```
char c = 'A';
```

byte、short、int、longは符号付き整数です。通常intを利用します。バイナリデータを扱う時はbyte、intで表現しきれない大きな桁を保持する時はlongを使います。shortはあまり利用しません。

float、doubleは浮動小数点数です。小数を使う時は通常floatを利用します。floatで表現しきれない大きな桁を保持する時はdoubleを使います。

booleanはtrue（真）、false（偽）のどちらかを格納したい時に使用します。代入する時は、そのまま「true」「false」と記述します。

```
boolean flag = true;
```

オブジェクト型の変数には、オブジェクトがどこにあるかというオブジェクトの参照を代入します。定義のみのオブジェクト型の変数は、初期値としてnullを保持しています。nullは「参照なし」という意味で、nullを保持する変数のフィールドやメソッドにアクセスするとエラーとなります。

Column» 配列

配列とは、1つの変数名と「添字」と呼ばれる数字を使って、ひとまとまりのデータを扱えるようにしたものです。配列を作成するための書式は次の通りです。

```
データ型[] 配列名 = new データ型[要素数];
```

たとえば3つの名前をそれぞれ変数に代入したとします。この時、変数が3つ必要になります。

```
name0 = "そらみ";
name1 = "へにへに";
name2 = "うのみ";
```

それに対して、配列を使うと1つの変数に3つ分の名前を入れることができます。変数名の後の[]の中に添字という番号を指定することによって、どの要素にアクセスするかを指定します。添字は「0」からはじまり、2番目の要素は「1」、3番目の要素は「2」というように指定します。

```
String[] name = new String[3];
name[0] = "そらみ";
name[1] = "へにへに";
name[2] = "うのみ";
```

ループを使うことにより、配列の全ての要素に処理を行うことができます。

```
for (int i = 0; i < name.length; i++) {
    name[i] += "さま";
}
```

3-4 イメージの描画



Star
http://book.mycom.co.jp/support/pc/star/34s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/34d.html

イメージファイルを読み込んで、**イメージを描画**するプログラムを作ります。



今回のプログラムは、次の2つのクラスで構成されています。

- **ImageExクラス（ImageEx.java）**
- **ImageCanvasクラス（ImageCanvas.java）**

プロジェクト名「**ImageEx**」でプロジェクトを作成してください。

▶イメージファイルの準備

今回使用するイメージファイルは次の1枚です。プロジェクトの「**res**」**フォルダ**に置いてください。「res」フォルダに配置したリソースはビルド時にJARファイルに含まれます。

・イメージ **jyagi.gif（48x48ドット）**



iアプリで使用できるイメージのファイル形式は「GIF」と「JPEG」です。イラストはGIF、写真や自然画風のイメージはJPEGを使うと、ファイルサイズを小さくてすみます。透過色が必要な時はGIFを使います。

Column» 急にJARファイルのサイズが大きくなる

Windowsで開発していると、JARファイルの中に「Thumbs.db」という名前のファイルが混ざり、無駄にサイズが大きくなることがあります。Thumbs.dbは、エクスプローラでフォルダ内のイメージファイルを「縮小」表示した時に作られるキャッシュファイルです。Windowsの初期設定では、Thumbs.dbは省略されて見えなくなっているので、表示するモードにしておきましょう。また、このファイルを作成しないように設定することもできます。

フォルダを表示しているウィンドウ上部にあるメニュー[ツール]-[フォルダオプション...]を選択し「表示」タブの中の「詳細設定」で「保護されたオペレーティングシステムファイルを表示しない（推奨）」のチェックを外し、[OK]ボタンをクリックします。また「縮小版キャッシュしない」にチェックを入れるとこのThumb.dbは作成されなくなります（008ページ参照）。

開発中のプロジェクトの「res」フォルダ内にThumb.dbがあった時は、削除してからビルドしなおしましょう。

▶ImageExクラス

ImageExクラスは、プログラムの本体となるクラスです。

Star ImageEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// イメージの描画 ( 本体 )
public class ImageEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {      // ⓒ
        Display.setCurrent(new ImageCanvas());
    }
}
```

DoJa ImageEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class ImageEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶ImageCanvasクラス

ImageCanvasクラスはキャンバスとなるクラスです。

Star ImageCanvas.java

```
import com.docomostar.media.MediaImage;   // ⓐ
import com.docomostar.media.MediaManager; //
import com.docomostar.ui.Canvas;          //
import com.docomostar.ui.Graphics;        //
import com.docomostar.ui.Image;           //

// イメージの描画(キャンバス)
public class ImageCanvas extends Canvas {
    private Image image = null;           // ⓑ イメージ

    // コンストラクタ ...❶
    public ImageCanvas() {
        try {
            // イメージファイルの読み込み ...❷
            MediaImage m;
            m = MediaManager.getImage("resource:///jyagi.gif");
            m.use();
            image = m.getImage();
        } catch (Exception e) {
            // 例外処理 ...❸
            System.out.println(e.toString());
        }
    }

    // 描画
    public void paint(Graphics g) {
        // イメージの描画 ...❹
```

Next page. ⬇

```
        g.drawImage(image,0,0);

        // 1/2 倍拡大 ...❺
        g.drawScaledImage(image,60,0,48/2,48/2,0,0,48,48);

        // 2 倍拡大
        g.drawScaledImage(image,120,0,48*2,48*2,0,0,48,48);

        // 左右反転 ...❻
        g.setFlipMode(Graphics.FLIP_HORIZONTAL);
        g.drawImage(image,0,120);

        // 上下反転
        g.setFlipMode(Graphics.FLIP_VERTICAL);
        g.drawImage(image,60,120);

        // 反転なし
        g.setFlipMode(Graphics.FLIP_NONE);
        g.drawImage(image,0,180);

        // 時計回りに 90 度回転
        g.setFlipMode(Graphics.FLIP_ROTATE_RIGHT);
        g.drawImage(image,60,180);

        // 時計回りに 180 度回転
        g.setFlipMode(Graphics.FLIP_ROTATE);
        g.drawImage(image,120,180);

        // 時計回りに 270 度回転
        g.setFlipMode(Graphics.FLIP_ROTATE_LEFT);
        g.drawImage(image,180,180);
    }
}
```

DoJa ImageCanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.MediaManager;
```

ⓑ

```
public class ImageCanvas extends Canvas {
    private Image image; // イメージ
```

ImageCanvas …❶：コンストラクタ

コンストラクタとは、オブジェクトを生成する時に利用するメソッドです。「new クラス名()」でクラスが生成された時に呼ばれます。メソッドの定義と異なり、戻り値の型はありません。

```
アクセス修飾子 クラス名 ( 引数の型1 引数名1, …) {
}
```

ImageCanvas …❷：イメージファイルの読み込み

イメージファイルを読み込むにはMediaManagerクラスのgetImage()メソッドを使います。

［MediaManagerクラス］

static MediaImage getImage(*location*)

［解説］ イメージファイルの読み込み
［引数］ *location* 読み込み先：String型
［戻り値］ MediaImageオブジェクト

引数の*location*（読み込み先）は次のフォーマットで指定します。

```
resource:/// ファイル名
```

戻り値としてMediaImageオブジェクトが返ってきます。MediaImageクラスは、ファイルを読み込み、内部表現形式に変換するクラスです。use()メソッドでイメージを利用可能にし、getImage()メソッドでImageクラスのオブジェクトを取得します。

［MediaImageインタフェース］

void use()

［解説］ イメージの利用可能化

［MediaImageインタフェース］

Image getImage()

［解説］ イメージオブジェクトの取得
［戻り値］ イメージオブジェクト

また、use()メソッドは処理に失敗した時に例外を投げるのでtry～catchで囲む必要があります。

ImageCanvas …❸：例外処理

例外は、プログラムの実行時に発生するエラーのことです。Javaでは例外が発生した時、どのような処理を行うかを、次の書式で記述します。

```
try {
    例外が発生するかもしれない処理
} catch ( 例外クラス名　変数名 ) {
    例外を捕まえた時の処理
}
```

tryブロックに「例外が発生するかもしれない処理」を記述し、catchブロックに「例外を捕まえた時の処理」を記述します。今回は、イメージファイルを読み込みに失敗した時、System.out.println()メソッドで、例外クラスの名前をiαppliToolのメッセージエリアに表示しています。

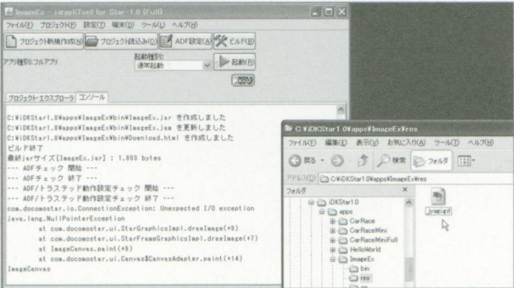
［ImageCanvas.javaの一部］

```
System.out.println(e.toString());
```

主な例外としては次のようなものがあります。

NullPointerException	オブジェクトのメソッドを呼んだ時に発生
ArrayIndexOutOfBoundsException	配列にマイナスかサイズ以上の添字が使われた時に発生
IOException	通信やデータ保存領域への入出力処理が失敗した時に発生
ArithmeticException	整数を0で除算した時に発生
NumberFormatException	文字列を数値に変換できない時に発生
IllegalArgumentException	不正な引数をメソッドに渡した時に発生
SecurityException	セキュリティ違反となる処理を行った時に発生
ClassNotFoundException	指定したクラスが見つからない時に発生

下記の画面ではファイル名が「_jyagi.gif」と変わってしまっているので、エラーNullPointerExceptionが発生しています。



ImageCanvas ...❹：イメージの描画

イメージの描画を行うには、GraphicsクラスのdrawImage()メソッドを使います。

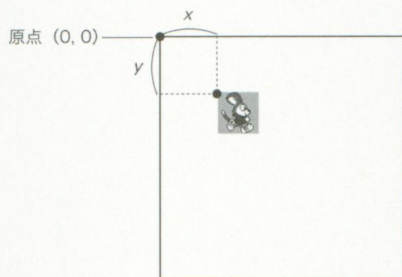
［Graphicsクラス］

void drawImage(*image, x, y*)

［解説］ イメージの描画

［引数］ *image* イメージオブジェクト：Image型
x X座標：int型
y Y座標：int型

引数の*x*と*y*（X、Y座標）はイメージの左上の座標です。



ImageCanvas ...❺：イメージの拡大縮小

イメージの拡大縮小を行うには、GraphicsクラスのdrawScaledImage()メソッドを使います。

［Graphicsクラス］

void drawScaledImage(*image, dx, dy, width, height, sx, sy, swidth, sheight*)

［解説］ イメージの拡大縮小

［引数］

image	イメージ：	Image型
dx	描画先の矩形のX座標：	int型
dy	描画先の矩形のY座標：	int型
width	描画先の矩形の幅：	int型
height	描画先の矩形の高さ：	int型
sx	描画元の矩形のX座標：	int型
sy	描画元の矩形のY座標：	int型
swidth	描画元の矩形の幅：	int型
sheight	描画元の矩形の高さ：	int型

ImageCanvas ...❻：イメージの反転

イメージの反転を行うには、GraphicsクラスのsetFlipMode()メソッドを使います。このメソッドを使用すると、それ以降のイメージを描画する時に指定された表示方法が適用されます。

［Graphicsクラス］

void setFlipMode(*flipmode*)

［解説］ イメージの反転

［引数］ *flipmode* 反転方法：int型

引数の*flipmode*（反転方法）には次の定数を指定します。

Graphics.FLIP_NONE	反転なし
Graphics.FLIP_HORIZONTAL	左右反転
Graphics.FLIP_VERTICAL	上下反転
Graphics.FLIP_ROTATE_RIGHT	時計回りに90度回転
Graphics.FLIP_ROTATE	時計回りに180度回転
Graphics.FLIP_ROTATE_LEFT	時計回りに270度回転

Column»　イメージデータの破棄

イメージファイルはいくらでも読み込めるわけではありません。オブジェクトを生成したり、イメージファイルやサウンドファイルを読み込むことによって、携帯電話内の**ヒープ**という作業領域が消費されます。そのため、作業領域がいっぱいになってしまうと、それ以上処理ができなくなり、「OutOfMemoryError」というエラーが発生してiアプリが終了してしまいます。

イメージデータが必要なくなったら破棄して、ヒープを空き状態に戻した方がよいでしょう。イメージを破棄するには、Imageクラスのdispose()メソッドを使います。オブジェクト参照にnullを指定しただけでは破棄されません。

［Imageクラス］

```
void dispose()
```

［解説］イメージの破棄

Column»　Javaヒープとネイティブデータヒープ

ヒープ容量は機種によって異なり、「Javaヒープ」と「ネイティブデータヒープ」の区別がある機種と、ない機種があります。JavaヒープはJavaのオブジェクトなどを格納するヒープ領域で、ネイティブデータヒープはメディアデータや画面表示のためのデータを格納するヒープ領域です。具体的な値は以下のサイトを参照してください。Star-1.0/DoJa-5.1プロファイル対応端末については本書の付録a-1（346ページ）に一覧を掲載しましたので、あわせて確認してください。

・iアプリ 端末スペック一覧
http://www.nttdocomo.co.jp/service/imode/make/content/spec/iappli/

3-5 アニメーションの表示

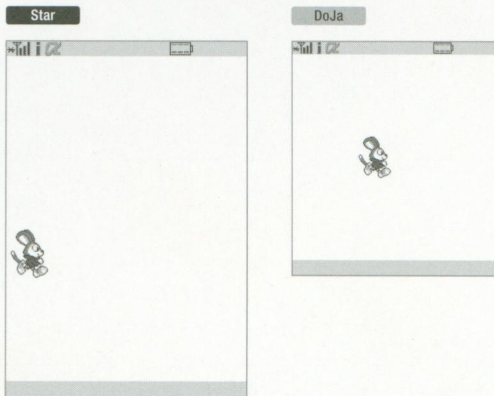


Star
http://book.mycom.co.jp/support/pc/star/35s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/35d.html

アニメーションの表示を行うプログラムを作ります。



今回のプログラムは、次の2つのクラスで構成されています。

- **AnimeEx**クラス（**AnimeEx.java**）
- **AnimeCanvas**クラス（**AnimeCanvas.java**）

プロジェクト名「**AnimeEx**」でプロジェクトを作成してください。

▶アニメーションファイルの準備

今回使用するアニメーションファイル（アニメーションGIFファイル）は次の1枚です。プロジェクトの「**res**」フォルダに置いてください

・アニメーションGIF **jyagi.gif**（**48x48**ドット）



▶AnimeExクラス

AnimeExクラスは、プログラムの本体となるクラスです。

Star AnimeEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// アニメーションの表示（本体）
public class AnimeEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {      // ⓒ
        AnimeCanvas c = new AnimeCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start(); // …❶
    }
}
```

DoJa AnimeEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class AnimeEx extends IApplication {
```

ⓒ

```
    public void start() {
```

AnimeEx …❶：メインループの実行

AnimeCanvasオブジェクトの生成後、AnimeCanvasクラスのexe()メソッドを呼んでいます。このexe()メソッドにはメインループが存在し、アプリ終了まで処理し続けます。

chapter

1
2
3
4
5
6
7
8
a

▶AnimeCanvasクラス

AnimeCanvasクラスは、キャンバスとなるクラスです。

Star AnimeCanvas.java

```
import com.docomostar.media.MediaImage;   // ⓐ
import com.docomostar.media.MediaManager; //
import com.docomostar.ui.Canvas;          //
import com.docomostar.ui.Graphics;        //
import com.docomostar.ui.Image;           //
// アニメーションの表示（キャンバス）
public class AnimeCanvas extends Canvas
    implements Runnable {
    private Graphics g;                  // グラフィックス
    private Image    image;              // イメージ
    private int      x  = getWidth()/2;  // X座標
    private int      y  = getHeight()/2; // Y座標
    private int      vx = 12;            // X速度
    private int      vy = 6;             // Y速度

    // 処理
    public void run() {
        // アニメーションファイルの読み込み …❶
        try {
            MediaImage m;
            m = MediaManager.getImage("resource:///jyagi.gif");
            m.use();
            image = m.getImage();
        } catch (Exception e) {
            System.out.println(e.toString());
        }

        // グラフィックスの取得 …❷
        g = getGraphics();

        // メインループ
        while (true) {
            // 移動
            x+ = vx;
            y+ = vy;
            if (x<0 || getWidth() <x) vx = -vx;
            if (y<0 || getHeight()<y) vy = -vy;

            // ダブルバッファリング …❸
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
```

```
                g.drawImage(image,x-24,y-24);
                g.unlock(true);

                // スリープ ...❹
                try {
                    Thread.sleep(100);
                } catch (Exception e) {
                }
            }
        }

    // 描画
    public void paint(Graphics g) {}
}
```

Star AnimeCanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.MediaManager;
```

AnimeCanvas ...❶：アニメーションファイルの読み込み

3.4節のImageCanvasクラスと同じように、アニメーションファイルを読み込むにはMediaManagerクラスのgetImage()メソッドを使います。

AnimeCanvas ...❷：Graphicsオブジェクトの取得

これまで描画処理をする時、paint()メソッドに渡されるGraphicsオブジェクトを利用してきましたが、CanvasクラスのgetGraphics()メソッドでGraphicsオブジェクトを取得することもできます。

［Canvasクラス］

Graphics getGraphics()

［解説］ Graphicsオブジェクトの取得
［戻り値］ Graphicsオブジェクト

AnimeCanvas ...❸：ダブルバッファリング

ダブルバッファリングとは、描画命令を実行ごとに画面に反映させるのではなく、複数の描画命令をまとめて画面に反映させることで、画面のチラつきを防ぐ処理のことです。使い方は、描画前にGraphicsクラスのlock()メソッド、描画後にunlock()メソッドを呼ぶだけです。unlock()メソッドの引数にはtrueを指定します。

［Graphicsクラス］

void lock()

［解説］ ロック

［Graphicsクラス］

void unlock(*forced*)

［解説］ アンロック

［引数］ *forced* 強制的に描画反映するか：boolean型

AnimeCanvas ...❹：スリープ

数ミリ秒間プログラムの処理を止めたい時は、Threadクラスのsleep()メソッドを使います。このメソッドは例外を投げるのでtryとcatchで囲む必要があります。例外を捕まえた時の処理は空で問題ありません。

［Threadクラス］

static void sleep(*time*)

［解説］ スリープ

［引数］ *time* スリープ時間（ミリ秒）：long型

100ミリ秒スリープするには、次のように記述します。

```
try {
    Thread.sleep(100);
} catch (Exception e) {
}
```

Column» 条件分岐

条件によって処理する内容を変える時は**if**、**switch**という条件分岐命令を使います。

if

ifは条件式の結果に応じて分岐を行う命令です。ifの「()」の中に条件式を指定し、その後ろのブロックに条件式の結果がtrueの時の処理を記述します。

［書式］

```
if ( 条件式 ) {
    処理
}
```

if ～ elseを使うと、条件式の結果がfalseの時の処理を記述します。

［書式］

```
if ( 条件式 ) {
    条件が成り立った時の処理
} else {
    条件が成り立たなかった時の処理
}
```

処理を3つ以上に分けたい時は、次のように記述できます。

［書式］

```
if ( 条件式 1) {
    条件式 1 が成り立った時の処理
} else if ( 条件式 2) {
    条件式 1 が成り立たず、条件式 2 が成り立った時の処理
} else {
    全ての条件式が成り立たなかった時の処理
}
```

switch

switchは変数の値に応じて分岐を行う命令です。switchの「()」の中に変数の値を指定し、その後ろのcaseに条件となる値と処理を記述します。

［書式］

```
switch( 式 ) {
    case 値1:
        式の値が値1の時の処理
        break;
    case 値2:
        式の値が値2の時の処理
        break;
    case 値3:
        式の値が値3の時の処理
        break;
    default:
        いずれの値にも一致しない時の処理
        break;
}
```

switchの「()」内の式の値は、int型、uint型、Number型のいずれかである必要があります。また、breakを入れない時は、次のcaseの処理を行います。どの条件にも該当しない時はdefaultに分岐します。

Column» ループ

条件によって処理する繰返す時はfor、whileというループ命令を使います。

for

forはループ回数が決まっている時にループを行う命令です。forの書式は次の通りです。

```
for ( 初期化式 ; 条件式 ; 増減式 ) {
    処理
}
```

たとえば1から1000までの和を求めたい時は次のように記述します。

```
int sum=0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
```

「for (int i = 1; i <= 1000; i++)」では、まず最初に「int i = 1」で変数iに1を代入しています。これをループの初期化式と呼びます。次の「i <= 1000」では、ループを繰り返す条件を指定しています。これは「iは1000以下」という意味になります。最後に記述した「i++」はiの値が1加算します。これはループの最後に毎回実行されます。

［書式］

```
for (int i=0; i<10; i++) {
    処理
}
```

while

whileは条件式がtrueになるまでループを行う命令です。whileの書式は次の通りです。

［書式］

```
while ( 条件式 ) {
    処理
}
```

whileによる1000以下の数値の和の計算プログラムは、次のように記述します。

```
int sum = 0;
int i = 1;
while (i <= 1000) {
    sum += i;
    i++;
}
```

3-6 キーイベントの処理

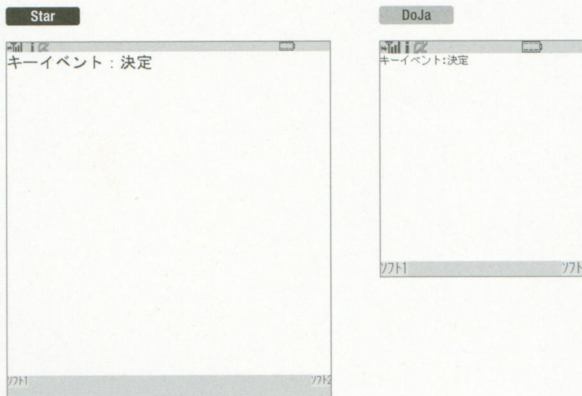


Star
http://book.mycom.co.jp/support/pc/star/36s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/36d.html

キーを押した時、**キーの名前**を画面に表示するプログラムを作ります。



今回のプログラムは、次の2つのクラスで構成されています。

- **KeyEventExクラス（KeyEventEx.java）**
- **KeyEventCanvasクラス（KeyEventCanvas.java）**

プロジェクト名「**KeyEventEx**」でプロジェクトを作成してください。

▶KeyEventExクラス

KeyEventExクラスは、プログラムの本体となるクラスです。

Star KeyEventEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //
```

```
// キーイベントの処理 ( 本体 )
public class KeyEventEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {         // ⓒ
        KeyEventCanvas c = new KeyEventCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa KeyEventEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class KeyEventEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶KeyEventCanvasクラス

KeyEventCanvasクラスは、キャンバスとなるクラスです。

Star KeyEventCanvas.java

```
import com.docomostar.ui.Canvas;   // ⓐ
import com.docomostar.ui.Display;  //
import com.docomostar.ui.Graphics; //

// キーイベントの処理 ( キャンバス )
public class KeyEventCanvas extends Canvas
    implements Runnable {
    private int keyEvent = -999; // キーイベント …❶

    // 実行
    public void run() {
        // グラフィック
        Graphics g = getGraphics();

        // 情報
        String info = "";
```

Next page.

```
        // ソフトキーとソフトラベル ...❷
        setSoftLabel(SOFT_KEY_1,"ソフト1"); // ⓑ
        setSoftLabel(SOFT_KEY_2,"ソフト2"); //

        while (true) {
            switch (keyEvent) {
            // 数字・記号キー
            case Display.KEY_0:        info = "0";   break;
            case Display.KEY_1:        info = "1";   break;
            case Display.KEY_2:        info = "2";   break;
            case Display.KEY_3:        info = "3";   break;
            case Display.KEY_4:        info = "4";   break;
            case Display.KEY_5:        info = "5";   break;
            case Display.KEY_6:        info = "6";   break;
            case Display.KEY_7:        info = "7";   break;
            case Display.KEY_8:        info = "8";   break;
            case Display.KEY_9:        info = "9";   break;
            case Display.KEY_ASTERISK: info = "*";   break;
            case Display.KEY_POUND:    info = "#";   break;
            // 方向・決定キー
            case Display.KEY_LEFT:     info = "左";  break;
            case Display.KEY_UP:       info = "上";  break;
            case Display.KEY_RIGHT:    info = "右";  break;
            case Display.KEY_DOWN:     info = "下";  break;
            case Display.KEY_SELECT:   info = "決定";break;
            // ソフトキー
            case Display.KEY_SOFT1:    info = "ソフト1";break; // ⓒ
            case Display.KEY_SOFT2:    info = "ソフト2";break; //
            }
            keyEvent = -999;

            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("キーイベント:"+info,0,24); // ⓓ
            g.unlock(true);

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // キーイベントの取得 ...❸
```

```
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa KeyEventCanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
```

ⓑ

```
        setSoftLabel(SOFT_KEY_1,"ｿﾌﾄ1");
        setSoftLabel(SOFT_KEY_2,"ｿﾌﾄ2");
```

ⓒ

```
            case Display.KEY_SOFT1:    info = "ｿﾌﾄ1";break;
            case Display.KEY_SOFT2:    info = "ｿﾌﾄ2";break;
```

ⓓ

```
            g.drawString("キーイベント："+info,0,12);
```

KeyEventCanvas ...❶：フィールド変数定義

フィールド変数とはオブジェクトの持つ変数で、オブジェクトが解放されるまで保持し続けます。さらにはオブジェクト内の全てのメソッドで利用できます。今回のプログラムではkeyEvent変数がフィールド変数で、exe()メソッドとprocessEvent()メソッドの両方からアクセスしています。

アクセス修飾子　変数の型　変数名；

KeyEventCanvas ...❷：ソフトキーとソフトラベル

画面下に表示されるメニューのことを**ソフトラベル**、そのメニューを実行するキーのことを**ソフトキー**と呼びます。ソフトキーには左下の「ソフトキー1」と右下の「ソフトキー2」があります。

ソフトラベルの文字列を指定するには、CanvasクラスのsetSoftLabel()メソッドを使います。DoJa指定のアプリケーションでは全角2字分のスペースしかなかったため、半角カナを使

うなどして表現していましたが、Starプロファイルからは全角8字分のスペースを利用できるようになりました。

[Canvasクラス]

void setSoftLabel(*key, caption*)

[解説] ソフトラベルの指定

[引数] *key* ソフトキーの種類:int型

caption 表示する文字列(全角2文字/半角4文字程度):String型

引数の*key*(ソフトキーの種類)には次の定数を指定します。

Canvas.SOFT_KEY_1	ソフトキー1
Canvas.SOFT_KEY_2	ソフトキー2

KeyEventCanvas ...❸:キーイベントの取得

CanvasクラスのprocessEvent()メソッドをオーバーライドすることによって、キーイベントを取得することができます。携帯電話のボタンが押されたり離されたりするたびに、実機からこのメソッドが呼ばれます。

[Canvasクラス]

void processEvent(*type, param*)

[解説] キャンバスでのイベント発生時に呼ばれる

[引数] *type* イベントの種類: int型

param パラメータ: int型

引数の*type*(イベントの種類)には次の定数が渡されます。**キープレスイベント**はボタンが押された時に発生するイベントで、**キーリリースイベント**はボタンが離された時に発生するイベントです。

Display.KEY_PRESSED_EVENT	キープレスイベント
Display.KEY_RELEASED_EVENT	キーリリースイベント

引数の*param*(パラメータ)には次の定数が渡されます。

Display.KEY_0	0キー
Display.KEY_1	1キー
Display.KEY_2	2キー
Display.KEY_3	3キー
Display.KEY_4	4キー
Display.KEY_5	5キー
Display.KEY_6	6キー
Display.KEY_7	7キー
Display.KEY_8	8キー
Display.KEY_9	9キー
Display.KEY_ASTERISK	*キー
Display.KEY_POUND	#キー
Display.KEY_SELECT	決定キー
Display.KEY_UP	上キー
Display.KEY_DOWN	下キー
Display.KEY_LEFT	左キー
Display.KEY_RIGHT	右キー
Display.KEY_SOFT1	ソフトキー1
Display.KEY_SOFT2	ソフトキー2

今回はキーを押した時、キーの種類を変数keyEventに代入しています。

KeyEventCanvas.javaの一部

```
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
```

keyEventの初期値の「-999」はキー定数に存在しない値として指定しています。

KeyEventCanvas.javaの一部

```
    private int keyEvent = -999; // キーイベント …❶
```

chapter 1 2 3 4 5 6 7 8 a

3-7 キー状態の処理

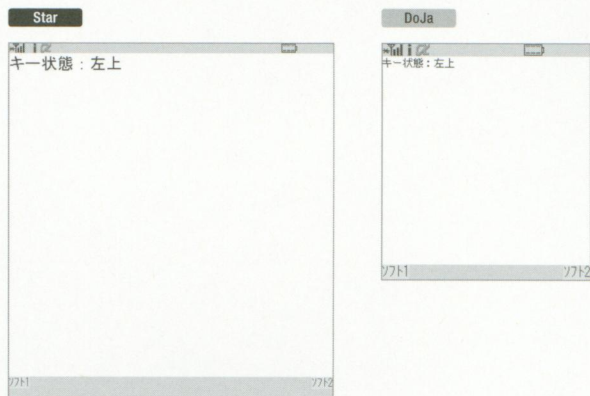


Star
http://book.mycom.co.jp/support/pc/star/37s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/37d.html

現在の**キー状態**を画面に表示するプログラムを作ります。キーイベント（3.6節参照）がボタンを押したり離したりするたびに処理を行うのに対し、キー状態は、現在どのキーが押されているかを取得して、それに応じた処理を行います。つまり、キーを押した直後に処理したい時はキーイベント、キーが押されている限り処理し続ける時はキー状態を使います。



今回のプログラムは、次の2つのクラスで構成されています。

- **KeyStateExクラス（KeyStateEx.java）**
- **KeyStateCanvasクラス（KeyStateCanvas.java）**

プロジェクト名「**KeyStateEx**」でプロジェクトを作成してください。

▶KeyStateExクラス

KeyStateExクラスは、プログラムの本体となるクラスです。

Star KeyStateEx.java

```
import com.docomostar.StarApplication; // ⓐ
```

```
import com.docomostar.ui.Display;      //

// キー状態の処理 ( 本体 )
public class KeyStateEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) { // ⓒ
        KeyStateCanvas c = new KeyStateCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa KeyStateEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class KeyStateEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶KeyStateCanvasクラス

KeyStateCanvasクラスは、キャンバスとなるクラスです

Star KeyStateCanvas.java

```
import com.docomostar.ui.Canvas;   // ⓐ
import com.docomostar.ui.Display;  //
import com.docomostar.ui.Graphics; //

// キー状態の処理 ( キャンバス )
public class KeyStateCanvas extends Canvas
    implements Runnable {

    // 処理
    public void run() {
        // グラフィック
        Graphics g = getGraphics();

        // キー状態
        int keyState = 0;
```

Next page. ⬇

```
// 情報
String info = "";

// ソフトラベル
setSoftLabel(SOFT_KEY_1,"ソフト1"); // ⓑ
setSoftLabel(SOFT_KEY_2,"ソフト2"); //

while (true) {
    // 情報の初期化
    info = "";

    // キー状態の取得 ...❶
    keyState = getKeypadState();

    // 数字・記号キー ...❷
    if (((1<<Display.KEY_0)       &keyState)! = 0) info += "0";
    if (((1<<Display.KEY_1)       &keyState)! = 0) info += "1";
    if (((1<<Display.KEY_2)       &keyState)! = 0) info += "2";
    if (((1<<Display.KEY_3)       &keyState)! = 0) info += "3";
    if (((1<<Display.KEY_4)       &keyState)! = 0) info += "4";
    if (((1<<Display.KEY_5)       &keyState)! = 0) info += "5";
    if (((1<<Display.KEY_6)       &keyState)! = 0) info += "6";
    if (((1<<Display.KEY_7)       &keyState)! = 0) info += "7";
    if (((1<<Display.KEY_8)       &keyState)! = 0) info += "8";
    if (((1<<Display.KEY_9)       &keyState)! = 0) info += "9";
    if (((1<<Display.KEY_ASTERISK)&keyState)! = 0) info += "*";
    if (((1<<Display.KEY_POUND)   &keyState)! = 0) info += "#";
    // 方向・決定キー
    if (((1<<Display.KEY_LEFT)    &keyState)! = 0) info += "左";
    if (((1<<Display.KEY_UP)      &keyState)! = 0) info += "上";
    if (((1<<Display.KEY_RIGHT)   &keyState)! = 0) info += "右";
    if (((1<<Display.KEY_DOWN)    &keyState)! = 0) info += "下";
    if (((1<<Display.KEY_SELECT)  &keyState)! = 0) info += "決定";
    // ソフトキー
    if (((1<<Display.KEY_SOFT1)   &keyState)! = 0) info += "ソフト1"; // ⓒ
    if (((1<<Display.KEY_SOFT2)   &keyState)! = 0) info += "ソフト2"; //

    // 画面の描画
    g.lock();
    g.setColor(g.getColorOfName(g.WHITE));
    g.fillRect(0,0,getWidth(),getHeight());
    g.setColor(g.getColorOfName(g.BLACK));
    g.drawString("キー状態:"+info,0,24) ); // ⓓ
    g.unlock(true);

    // スリープ
    try {
        Thread.sleep(100);
```

```
            } catch (Exception e) {
            }
        }
    }

    // 描画
    public void paint(Graphics g) {}
}
```


DoJa KeyStateCanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
```

ⓑ

```
        setSoftLabel(SOFT_KEY_1,"ｿﾌﾄ1");
        setSoftLabel(SOFT_KEY_2,"ｿﾌﾄ2");
```

ⓒ

```
            case Display.KEY_SOFT1:    info = "ｿﾌﾄ1";break;
            case Display.KEY_SOFT2:    info = "ｿﾌﾄ2";break;
```

ⓓ

```
            g.drawString("キー状態："+info,0,12);
```

KeyStateCanvas …❶：キー状態の取得

キー状態を取得するには、CanvasクラスのgetKeypadState()メソッドを使います。

［Canvasクラス］

int getKeypadState()

［解説］　キー状態の取得

［戻り値］　キー状態

KeyStateCanvas ...❷：キー状態の処理

getKeypadState()メソッドの戻り値には、現在押されているキーの「1<<キー定数」の論理和を保持しています。キー定数はDisplayクラスで保持しています。

キー定数	意味	値
Display.KEY_0	0キー	0
Display.KEY_1	1キー	1
Display.KEY_2	2キー	2
Display.KEY_3	3キー	3
Display.KEY_4	4キー	4
Display.KEY_5	5キー	5
Display.KEY_6	6キー	6
Display.KEY_7	7キー	7
Display.KEY_8	8キー	8
Display.KEY_9	9キー	9

キー定数	意味	値
Display.KEY_ASTERISK	*キー	10
Display.KEY_POUND	#キー	11
Display.KEY_LEFT	左キー	16
Display.KEY_UP	上キー	17
Display.KEY_RIGHT	右キー	18
Display.KEY_DOWN	下キー	19
Display.KEY_SELECT	選択キー	20
Display.KEY_SOFT1	ソフトキー 1	21
Display.KEY_SOFT2	ソフトキー 2	22

2キーを押しているかどうかは次のような条件式で判定できます。

```
if (((1<<Display.KEY_2)&keyState)!=0) {
    2 キー押し中の処理
} else {
    2 キー離し中の処理
}
```

2キーと3キーを同時に押しているかどうかは次のような条件式で判定します。

```
if (((1<<Display.KEY_2)&keyState)!=0 &&|
    ((1<<Display.KEY_3)&keyState)!=0) {
    2 キーと 3 キーの同時押し中の処理
} else {
    2 キーと 3 キーのどちらか離し中の処理
}
```

Column» 演算子

Javaの演算子としては「算術演算子」「関係演算子」「複合代入演算子」「論理演算子」「条件演算子」「シフト演算子」があります。

算術演算子

算術演算子とは「+」や「-」のように計算を行うための演算子のことです。加算、減算、乗算、除算の他に、剰余算があります。剰余算は除算の余りを求めるためのものです。

演算子	意味	例	説明
+	加算	a + b	aとbを足す
-	減算	a - b	aからbを引く
*	乗算	a * b	aとbを掛ける
/	除算	a / b	aをbで割る
%	剰余算	a % b	aをbで割った余りを求める

演算子には優先順位があり、「乗算・除算」は「加算・減算」より優先されます。優先順位を変更するにはカッコ「()」を使います。

関係演算子

関係演算子とは2つの値を比較して、結果をtrueまたはfalseの値を返す演算子です。

演算子	説明	例
==	等しい	a == b
!=	等しくない	a != b
>	より大きい	a > b
>=	以上	a >= b
<	より小さい	a < b
<=	以下	a <= b

複合代入演算子

複合代入演算子とは代入と一緒に他の演算を行える演算子のことです。
「a = a+10;」も代入演算子を使えば、「a += 10;」と短くなります。

演算子	使い方	説明
+=	a += 10	a = a + 10と同じ
-=	a -= 10	a = a - 10と同じ
*=	a *= 10	a = a * 10と同じ
/=	a /= 10	a = a / 10と同じ
%=	a %= 10	a = a % 10と同じ

chapter
1
2
3
4
5
6
7
8
a

論理演算子

論理演算子とは、条件を否定したり、複数の条件を組み合わせることができる演算子です。boolean型の値に対して演算を行うことができます。論理演算子をifの条件式の中に記述することで「条件式aが成り立たない場合」や「条件aが成り立ち、かつ条件bも成り立つ場合」にブロック内の処理を実行するといったことができます。

演算子	説明	例
!	aがtrueであればfalse、falseであればtrue	!a
&&	aとbの両方がtrueであるならtrue、そうでなければfalse	a && b
||	aとbのどちらかがtrueであるならtrue、そうでなければfalse	a || b

条件演算子

条件演算子とは、ifと同じような働きをする演算子です。条件演算子の書式は次の通りです。

```
条件式 ? 値1 : 値2
```

条件式が成立した時は値1、条件式が成立しなかった時は値2を返します。
2つの数値のうち大きい値を取得したい時は、次のように記述します。

```
max = (price1>price2) ? price1:price2;
```

シフト演算子

シフト演算子とはビットを左右にシフトさせる演算子のことです。

演算子	例	説明
<<	x << y	xのビットをyビットだけ左にシフトさせる 右端には0を埋める
>>	x >> y	xのビットをyビットだけ右にシフトさせる 左端にはその時の最上位ビットと同じ符号を埋める
>>	x >>> y	xのビットをyビットだけ、右にシフトさせる 左端には0を埋める

3-8 サウンドの再生



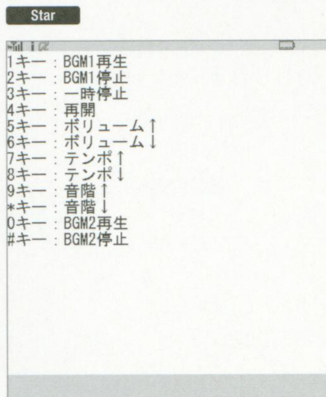
Star
http://book.mycom.co.jp/support/pc/star/38s.html



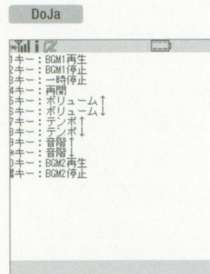
DoJa
http://book.mycom.co.jp/support/pc/star/d/38d.html

サウンドを再生するプログラムを作ります。

Star



DoJa



1キー	BGM1再生
2キー	BGM1停止
3キー	一時停止
4キー	再開
5キー	ボリューム↑
6キー	ボリューム↓

7キー	テンポ↑
8キー	テンポ↓
9キー	音階↑
*キー	音階↓
0キー	BGM2再生
#キー	BGM2停止

「マナーモード」時、「アプリ音量」(着信音量と連動している端末もあります)が小さい時、「ボタン確認音」がオンの時は、音が再生されないことがあります。エミュレータでも正しく音が再生されないことがあります。

このプログラムは、次の2つのクラスで構成されています。

- **SoundsExクラス（SoundsEx.java）**
- **SoundsCamvasクラス（SoundsCamvas.java）**

プロジェクト名「**SoundsEx**」でプロジェクトを作成してください。

▶サウンドファイルの準備

今回使用する**サウンドファイル**は次の2つです。プロジェクトの「**res**」**フォルダ**に置いてください。iアプリで利用可能なサウンドファイル形式は**MLDファイル**です。iモードの着メロとして使われています。

- BGM：**0.mld**
- 効果音：**1.mld**

▶SoundsExクラス

SoundsExクラスは、プログラムの本体となるクラスです。

Star SoundsEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// サウンドを再生する(本体)
public class SoundEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {      // ⓒ
        SoundCanvas c = new SoundCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa SoundsEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class SoundEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶SoundsCanvasクラス

SoundsCanvasクラスは、キャンバスとなるクラスです。

Star SoundsCanvas.java

```
import com.docomostar.ui.AudioPresenter;  // ⓐ
import com.docomostar.ui.Canvas;          //
import com.docomostar.ui.Display;         //
import com.docomostar.ui.Graphics;        //
import com.docomostar.ui.MediaListener;   //
import com.docomostar.ui.MediaPresenter;  //
import com.docomostar.media.MediaManager; //
import com.docomostar.media.MediaSound;   //

// サウンドの再生(キャンバス)
public class SoundCanvas extends Canvas
    implements Runnable,MediaListener { // …❸-1
    private int keyEvent = -999;        // キーイベント
    private int volume   = 100;         // ボリューム
    private int tempo    = 100;         // テンポ
    private int trans    = 0;           // 音階
    private AudioPresenter[] player = new AudioPresenter[2];

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        for (int i = 0;i<2;i++) {
            try {
                // サウンドファイルの読み込み …❶
                MediaSound sound = MediaManager.getSound(
                    "resource:///"+i+".mld");
                sound.use();

                // オーディオプレゼンタの生成 …❷
                player[i] = AudioPresenter.getAudioPresenter(i);
                player[i].setSound(sound);

                // メディアリスナの指定 …❸-2
                player[i].setMediaListener(this);
            } catch (Exception e) {
            }
        }

        while (true) {
            // 画面の描画
```

Next page. ⬇

```
g.lock();
g.setColor(g.getColorOfName(g.WHITE));
g.fillRect(0,0,getWidth(),getHeight());
g.setColor(g.getColorOfName(g.BLACK));
g.drawString("1キー：BGM1再生",    0,24*1);  // ⓑ
g.drawString("2キー：BGM1停止",    0,24*2);  //
g.drawString("3キー：一時停止",    0,24*3);  //
g.drawString("4キー：再開",        0,24*4);  //
g.drawString("5キー：ボリューム↑", 0,24*5);  //
g.drawString("6キー：ボリューム↓", 0,24*6);  //
g.drawString("7キー：テンポ↑",     0,24*7);  //
g.drawString("8キー：テンポ↓",     0,24*8);  //
g.drawString("9キー：音階↑",       0,24*9);  //
g.drawString("*キー：音階↓",       0,24*10); //
g.drawString("0キー：BGM2再生",    0,24*11); //
g.drawString("#キー：BGM2停止",    0,24*12); //
g.unlock(true);

// オーディオプレゼンタの操作 ...❹
if (keyEvent==Display.KEY_1) {
    // BGM1再生
    player[0].play();
} else if (keyEvent==Display.KEY_2) {
    // BGM1停止
    player[0].stop();
} else if (keyEvent==Display.KEY_3) {
    // 一時停止
    player[0].pause();
} else if (keyEvent==Display.KEY_4) {
    // 再開
    player[0].restart();
} else if (keyEvent==Display.KEY_0) {
    // BGM2再生
    player[1].play();
} else if (keyEvent==Display.KEY_POUND) {
    // BGM2停止
    player[1].stop();
}

// サウンドエフェクトの指定 ...❺
else if (keyEvent==Display.KEY_5) {
    // ボリューム↑
    volume += 10;
    if (volume>100) volume = 100;
    player[0].setAttribute(
            AudioPresenter.SET_VOLUME,volume);
} else if (keyEvent==Display.KEY_6) {
    // ボリューム↓
    volume -= 10;
```

```
                if (volume<0) volume = 0;
                player[0].setAttribute(AudioPresenter.SET_VOLUME,volume);
            } else if (keyEvent==Display.KEY_7) {
                // テンポ↑
                tempo += 10;
                if (tempo>400) tempo = 400;
                player[0].setAttribute(AudioPresenter.CHANGE_TEMPO,tempo);
            } else if (keyEvent==Display.KEY_8) {
                // テンポ↓
                tempo -= 10;
                if (tempo<30) tempo = 30;
                player[0].setAttribute(AudioPresenter.CHANGE_TEMPO,tempo);
            } else if (keyEvent==Display.KEY_9) {
                // 音階↑
                trans += 1;
                player[0].setAttribute(AudioPresenter.TRANSPOSE_KEY,trans);
            } else if (keyEvent==Display.KEY_ASTERISK) {
                // 音階↓
                trans -= 1;
                player[0].setAttribute(AudioPresenter.TRANSPOSE_KEY,trans);
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // メディアリスナのイベント処理 ...❸-3
    public void mediaAction(MediaPresenter source,int type,int param) {
        // ループ
        if (type==AudioPresenter.AUDIO_COMPLETE) {
            player[0].play();
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

Next page. ↓

DoJa SoundsCanvas.java

ⓐ

```
import com.nttdocomo.ui.AudioPresenter;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.MediaListener;
import com.nttdocomo.ui.MediaManager;
import com.nttdocomo.ui.MediaPresenter;
import com.nttdocomo.ui.MediaSound;
```

ⓑ

```
            g.drawString("1キー：BGM1再生",    0,12*1);
            g.drawString("2キー：BGM1停止",    0,12*2);
            // ...
            g.drawString("#キー：BGM2停止",    0,12*12);
```

SoundsCanvas ...❶：サウンドファイルの読み込み

サウンドファイルを読み込むにはMediaManagerクラスのgetSound()メソッドを使います。

［MediaManagerクラス］

static MediaSound getSound(*location*)

［解説］ サウンドファイルの読み込み
［引数］ *location* 読み込み先：String型
［戻り値］ MediaSoundオブジェクト

引数の*location*（読み込み先）は次のフォーマットで指定します。

```
resource:/// ファイル名
```

戻り値としてMediaSoundオブジェクトが返ってきます。MediaSoundインタフェースはサウンドファイルを読み込み、内部表現形式に変換するクラスです。MediaSoundインタフェースのuse()メソッドでサウンドデータの有効化を行います。

［MediaSoundインタフェース］

void use()

［解説］ サウンドデータの有効化

SoundsCanvas ...❷：オーディオプレゼンタの生成

読み込んだサウンドデータを再生するには、AudioPresenterクラスを使います。AudioPresenterクラスのgetAudioPresenter()メソッドでAudioPresenterオブジェクトを取得します。

［AudioPresenterクラス］

static AudioPresenter getAudioPresenter(*track*)

［解説］　オーディオプレゼンタの生成
［引数］　*track*　トラック番号（0～3）：int型
［戻り値］　AudioPresenterオブジェクト

引数の*track*（トラック番号）の範囲は機種依存ですが、ほぼ全ての端末の同時発音数が4（トラック番号は0～3）です。

オーディオプレゼンタにサウンドデータを指定するには、AudioPresenterクラスのsetSound()メソッドを使います。

［AudioPresenterクラス］

void setSound(*sound*)

［解説］　サウンドデータの指定
［引数］　*sound*　サウンドデータ：MediaSound型

SoundsCanvas ...❸：メディアリスナの指定

演奏の開始や完了などの演奏イベントの発生を知るには、MediaListenerインタフェースを使います。MediaListenerインタフェースを使うにはまず、SoundCanvasクラス自身にインタフェースを実装します。Javaでは、インタフェースを実装していることを示すために、implementsの後にインタフェース名を記述します（❸-1）。

次に、AudioPresenterクラスのsetMediaListener()メソッドで、イベントが発生した時、どこへ伝えればよいかを指定します。今回はSoundCanvasクラス自身がリスナになるのでthisをセットします（❸-2）。

［AudioPresenterクラス］

void setMediaListener(*listener*)

［解説］　メディアリスナの指定
［引数］　*listener*　メディアリスナ：MediaListener型

最後にmediaAction()メソッドを実装します。mediaAction()メソッドはメディアリスナイベントの発生時に呼ばれます（❸-3）。

［MediaListenerインタフェース］

void mediaAction(*source, type, param*)

［解説］ メディアリスナイベントの発生時に呼ばれる
［引数］ *source* 発生元のメディアプレゼンタ：MediaPresenter型
type イベント種別：int型
param パラメータ： int型

引数の*type*（イベント種別）には次の定数が渡されます。

AudioPresenter.AUDIO_PLAYING	演奏の開始
AudioPresenter.AUDIO_STOPPED	演奏の停止
AudioPresenter.AUDIO_COMPLETE	演奏の完了

第2引数の*param*（パラメータ）は使いません。

SoundsCanvas ...❹：オーディオプレゼンタの操作

オーディオプレゼンタに指定したサウンドを再生するにはplay()メソッド、停止するにはstop()メソッド、一時停止するにはpause()メソッド、再開するにはrestart()メソッドを使います。

［AudioPresenterクラス］

void play()

［解説］ サウンドの再生

［AudioPresenterクラス］

void stop()

［解説］ サウンドの停止

［AudioPresenterクラス］

void pause()

［解説］ サウンドの一時停止

［AudioPresenterクラス］

void restart()

［解説］ サウンドの再開

SoundsCanvas ...❺：サウンドエフェクトの指定

iアプリで設定可能なサウンドエフェクトには**ボリューム**、**テンポ**、**音階**があります。ボリュームは音の大きさで、指定なしの時に再生される音量に対して、実際に再生する音量のパーセンテージ（0〜100）を指定します。テンポは音の速さで、指定なしの時に再生される速さに対して、実際に再生する速さのパーセンテージ（25〜400）を指定します。音階は音の高さで、半音階単位で音階の増減を指定します。

サウンドエフェクトを指定するには、AudioPresenterクラスのsetAttribute()メソッドを使います。

［AudioPresenterクラス］

void setAttribute(*attr*, *value*)

［解説］ サウンドエフェクトの指定

［引数］ *attr* エフェクトの種類：int型

value 値：int型

引数の*attr*（エフェクトの種類）には次の定数と値を指定します。

エフェクトの種類	説明	初期値	値の範囲
AudioPresenter.SET_VOLUME	ボリューム	100	0〜100（%）
AudioPresenter.CHANGE_TEMPO	テンポ	100	25〜400（%）
AudioPresenter.TRANSPOSE_KEY	音階	0	機種依存（半音階単位）

Column» サウンドデータの破棄

サウンドデータが必要なくなったら破棄して、**ヒープ**を空き状態に戻した方がよいでしょう。サウンドデータを破棄するには、`MediaSound`クラスの`dispose()`メソッドを使います。オブジェクト参照に`null`を指定しただけでは破棄されません。

［MediaSoundクラス］

```
void dispose()
```

［解説］メディアリソースの破棄

ネイティブアプリケーション連携

4

本章では、カメラやコードリーダなどの携帯電話の機能を利用したり、電波状況や電池残量などの携帯電話の情報を取得したりする方法について解説します。

4-1 端末情報の取得



Star
http://book.mycom.co.jp/support/pc/star/41s.html



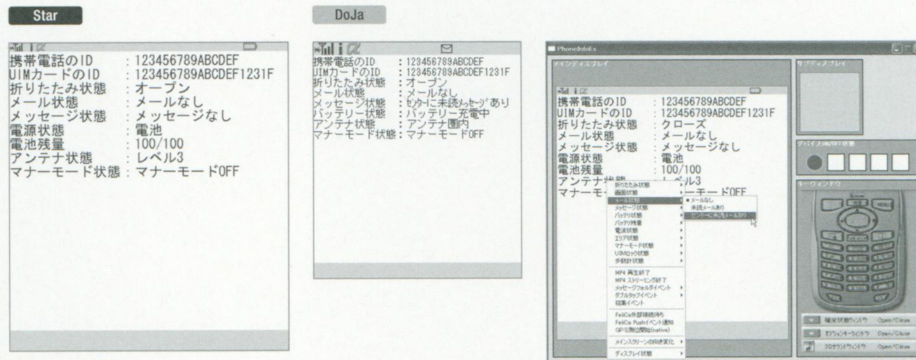
DoJa
http://book.mycom.co.jp/support/pc/star/d/41d.html

端末情報を表示するプログラムを作ります。取得する情報は次の8種類です。

- 携帯電話のID
- UIMカードのID
- 折りたたみ状態
- メール状態
- メッセージ状態
- バッテリー状態
- アンテナ状態
- マナーモード状態

「メール状態」「メッセージ状態」「バッテリー状態」「アンテナ状態」「マナーモード状態」はユーザが端末設定で「アイコン情報-利用する」を選択しておく必要があります。

エミュレータでは右クリックで開く「ポップアップメニュー」で端末状態を指定することができます。



このプログラムは、次の2つのクラスで構成されています。

- **PhoneInfoExクラス（PhoneInfoEx.java）**
- **PhoneInfoCanvasクラス（PhoneInfoCanvas.java）**

プロジェクト名「**PhoneInfoEx**」でプロジェクトを作成してください。

▶PhoneInfoExクラス

PhoneInfoExクラスは、プログラムの本体となるクラスです。

Star PhoneInfoEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// 端末情報の取得（本体）
public class PhoneInfoEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {          // ⓒ
        PhoneInfoCanvas c = new PhoneInfoCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa PhoneInfoEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class PhoneInfoEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶PhoneInfoCanvasクラス

PhoneInfoCanvasクラスはキャンバスとなるクラスです。

Star PhoneInfoCanvas.java

```
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Graphics;
import com.docomostar.system.PhoneSystem;

// 端末情報の取得（キャンバス）
public class PhoneInfoCanvas extends Canvas
    implements Runnable {
```

```
// 処理
public void run() {
    // グラフィックス
    Graphics g = getGraphics();

    while (true) {
        // 画面の描画
        g.lock();
        g.setColor(g.getColorOfName(g.WHITE));
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(g.getColorOfName(g.BLACK));

        // 個体識別情報の取得 ...❶
        g.drawString("携帯電話のID     :"+
            PhoneSystem.getProperty(PhoneSystem.TERMINAL_ID),0,24*1);
        g.drawString("UIMカードのID   :"+
            PhoneSystem.getProperty(PhoneSystem.USER_ID),0,24*2);

        // 端末情報の取得 ...❷
        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_FOLDING)) {
        case PhoneSystem.ATTR_FOLDING_OPEN:
            g.drawString("折りたたみ状態  :オープン",0,24*3);
            break;
        case PhoneSystem.ATTR_FOLDING_CLOSE:
            g.drawString("折りたたみ状態  :クローズ",0,24*3);
            break;
        }

        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_MAILBOX)) {
        case PhoneSystem.ATTR_MAIL_NONE:
            g.drawString("メール状態      :メールなし",0,24*4);
            break;
        case PhoneSystem.ATTR_MAIL_RECEIVED:
            g.drawString("メール状態      :未読メールあり",0,24*4);
            break;
        case PhoneSystem.ATTR_MAIL_AT_CENTER:
            g.drawString("メール状態      :センターに未読メールあり",0,24*4);
            break;
        }

        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_MESSAGEBOX)) {
        case PhoneSystem.ATTR_MESSAGE_NONE:
            g.drawString("メッセージ状態  :メッセージなし",0,24*5);
            break;
        case PhoneSystem.ATTR_MESSAGE_RECEIVED:
            g.drawString("メッセージ状態  :未読メッセージあり",0,24*5);
            break;
        case PhoneSystem.ATTR_MESSAGE_AT_CENTER:
```

Next page. ⬇

```
        g.drawString("メッセージ状態  :センターに未読メッセージあり",0,24*5);
        break;
    }

    switch (PhoneSystem.getAttribute(PhoneSystem.DEV_POWER_SOURCE)) {
    case PhoneSystem.ATTR_POWER_BATTERY:
        g.drawString("電源状態        :電池",0,24*6);
        break;
    case PhoneSystem.ATTR_POWER_EXTERNAL:
        g.drawString("電源状態        :外部電源",0,24*6);
        break;
    }

    int battery = PhoneSystem.getAttribute(PhoneSystem.DEV_BATTERY_LEVEL);
    int max     = PhoneSystem.getAttribute(PhoneSystem.DEV_MAX_BATTERY_LEVEL);
        g.drawString("電池残量        :"+battery+"/"+max,0,24*7);

    switch (PhoneSystem.getAttribute(PhoneSystem.DEV_SIGNAL_STATE)) {
    case PhoneSystem.ATTR_SIGNAL_LEVEL_1:
        g.drawString("アンテナ状態    :レベル1",0,24*8);
        break;
    case PhoneSystem.ATTR_SIGNAL_LEVEL_2:
        g.drawString("アンテナ状態    :レベル2",0,24*8);
        break;
    case PhoneSystem.ATTR_SIGNAL_LEVEL_3:
        g.drawString("アンテナ状態    :レベル3",0,24*8);
        break;
    case PhoneSystem.ATTR_SIGNAL_OUTSIDE:
        g.drawString("アンテナ状態    :圏外",0,24*8);
        break;
    }

    switch (PhoneSystem.getAttribute(PhoneSystem.DEV_MANNER)) {
    case PhoneSystem.ATTR_MANNER_ON:
        g.drawString("マナーモード状態:マナーモードON",0,24*9);
        break;
    case PhoneSystem.ATTR_MANNER_OFF:
        g.drawString("マナーモード状態:マナーモードOFF",0,24*9);
        break;
    }

    // 画面に反映
    g.unlock(true);

    // スリープ
    try {
        Thread.sleep(100);
    } catch (Exception e) {
    }
```

```
        }
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa PhoneInfoCanvas.java

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.PhoneSystem;
import com.nttdocomo.util.Phone;

// 端末情報の取得（キャンバス）
public class PhoneInfoCanvas extends Canvas
    implements Runnable {

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));

            // 個体識別情報の取得 ...❶
            g.drawString("携帯電話のID     ："+
                Phone.getProperty(Phone.TERMINAL_ID),0,12);
            g.drawString("UIMカードのID   ："+
                Phone.getProperty(Phone.USER_ID),0,24);

            // 端末情報の取得 ...❷
            switch (PhoneSystem.getAttribute(PhoneSystem.DEV_FOLDING)) {
            case PhoneSystem.ATTR_FOLDING_OPEN:
                g.drawString("折りたたみ状態　：オープン",0,36);
                break;
            case PhoneSystem.ATTR_FOLDING_CLOSE:
                g.drawString("折りたたみ状態　：クローズ",0,36);
                break;
            }

            switch (PhoneSystem.getAttribute(PhoneSystem.DEV_MAILBOX)) {
            case PhoneSystem.ATTR_MAIL_NONE:
```

Next page. ⬇

```
            g.drawString("メール状態       :メールなし",0,48);
            break;
        case PhoneSystem.ATTR_MAIL_RECEIVED:
            g.drawString("メール状態       :未読メールあり",0,48);
            break;
        case PhoneSystem.ATTR_MAIL_AT_CENTER:
            g.drawString("メール状態       :ｾﾝﾀｰに未読メールあり",0,48);
            break;
        }

        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_MESSAGEBOX)) {
        case PhoneSystem.ATTR_MESSAGE_NONE:
            g.drawString("メッセージ状態  :メッセージなし",0,60);
            break;
        case PhoneSystem.ATTR_MESSAGE_RECEIVED:
            g.drawString("メッセージ状態  :未読メッセージあり",0,60);
            break;
        case PhoneSystem.ATTR_MESSAGE_AT_CENTER:
            g.drawString("メッセージ状態  :ｾﾝﾀｰに未読ﾒｯｾｰｼﾞあり",0,60);
            break;
        }

        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_BATTERY)) {
        case PhoneSystem.ATTR_BATTERY_CHARGING:
            g.drawString("バッテリー状態  :バッテリー充電中",0,72);
            break;
        case PhoneSystem.ATTR_BATTERY_FULL:
            g.drawString("バッテリー状態  :バッテリーMAX",0,72);
            break;
        case PhoneSystem.ATTR_BATTERY_PARTIAL:
            g.drawString("バッテリー状態  :バッテリーその他",0,72);
            break;
        }

        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_SERVICEAREA)) {
        case PhoneSystem.ATTR_SERVICEAREA_OUTSIDE:
            g.drawString("アンテナ状態     :アンテナ圏外",0,84);
            break;
        case PhoneSystem.ATTR_SERVICEAREA_INSIDE:
            g.drawString("アンテナ状態     :アンテナ圏内",0,84);
            break;
        }

        switch (PhoneSystem.getAttribute(PhoneSystem.DEV_MANNER)) {
        case PhoneSystem.ATTR_MANNER_ON:
            g.drawString("マナーモード状態:マナーモードON",0,96);
            break;
        case PhoneSystem.ATTR_MANNER_OFF:
            g.drawString("マナーモード状態:マナーモードOFF",0,96);
```

```
                    break;
                }

                // 画面に反映
                g.unlock(true);

                // スリープ
                try {
                    Thread.sleep(100);
                } catch (Exception e) {
                }
            }
        }

    // 描画
    public void paint(Graphics g) {}
}
```


PhoneInfoCanvas ...❶：個体識別情報の取得

Star/DoJa

携帯電話のIDは携帯電話のハードを一意に識別するためのIDで、**UIMカードのID**は携帯電話のユーザを一意に識別するためのIDです。UIMカードはユーザの識別に利用する着脱可能な小さなカードのことです。機種変更時にはこれを新しい端末に差し替えるだけで手続きが完了します。ユーザが複数の端末を用途に合わせて使い分けできるという利点もあります。

Star

個体識別情報を取得するには、PhoneSystemクラスのgetProperty()メソッドを使います。

Star ［PhoneSystemクラス］

static int getProperty(*attr*)

［解説］ 個体識別情報の取得
［引数］ *attr* 属性：int型
［戻り値］ 個体識別情報

引数の*attr*（属性）には次の定数を指定します。

PhoneSystem.TERMINAL_ID	携帯電話のID
PhoneSystem.USER_ID	UIMカードのID

DoJa

DoJa対応端末の場合、個体識別情報を取得するには、PhoneクラスのgetProperty()メソッドを使います。

DoJa ［Phoneクラス］

static int getProperty(*attr*)

［解説］ 個体識別情報の取得
［引数］ *attr* 属性：int型
［戻り値］ 個体識別情報

引数の*attr*（属性）には次の定数を指定します。

Phone.TERMINAL_ID	携帯電話のID
Phone.USER_ID	UIMカードのID

PhoneInfoCanvas ...❷：端末情報の取得

Star/DoJa

個体識別情報以外の端末情報を取得するには、PhoneSystemクラスのgetAttribute()メソッドを使います。

［PhoneSystemクラス］

static int getAttribute(*attr*)

［解説］ 端末情報の取得
［引数］ *attr* 属性：int型
［戻り値］ 端末情報

引数の*attr*（属性）にはPhoneSystemクラスの持つ定数を指定します。戻り値にもPhoneSystemクラスの持つ定数が返ってきます。Starプロファイルの電池残量と最大電池残量は値が返ってきます。

Star

属性	戻り値	
PhoneSystem.DEV_FOLDING（折りたたみ状態）	PhoneSystem.ATTR_FOLDING_OPEN	オープン
	PhoneSystem.ATTR_FOLDING_CLOSE	クローズ
PhoneSystem.DEV_MAILBOX（メール状態）	PhoneSystem.ATTR_MAIL_NONE	メールなし
	PhoneSystem.ATTR_MAIL_RECEIVED	未読メールあり
	PhoneSystem.ATTR_MAIL_AT_CENTER	センターに未読メールあり
PhoneSystem.DEV_MESSAGEBOX（メッセージ状態）	PhoneSystem.ATTR_MESSAGE_NONE	メッセージなし
	PhoneSystem.ATTR_MESSAGE_RECEIVED	未読メッセージあり
	PhoneSystem.ATTR_MESSAGE_AT_CENTER	センターに未読メッセージあり
PhoneSystem.DEV_POWER_SOURCE（電源）	PhoneSystem.ATTR_POWER_BATTERY	電池
	PhoneSystem.ATTR_POWER_EXTERNAL	外部電源
	PhoneSystem.DEV_BATTERY_LEVEL	電池残量
	PhoneSystem.DEV_MAX_BATTERY_LEVEL	最大電池残量
PhoneSystem.DEV_SIGNAL_STATE（アンテナ状態）	PhoneSystem.ATTR_SIGNAL_LEVEL_1	レベル1
	PhoneSystem.ATTR_SIGNAL_LEVEL_2	レベル2
	PhoneSystem.ATTR_SIGNAL_LEVEL_3	レベル3
	PhoneSystem.ATTR_SIGNAL_OUTSIDE	圏外
PhoneSystem.DEV_MANNER（マナーモード状態）	PhoneSystem.ATTR_MANNER_ON	マナーモードON
	PhoneSystem.ATTR_MANNER_OFF	マナーモードOFF

DoJa

属性	戻り値	
PhoneSystem.DEV_FOLDING（折りたたみ状態）	PhoneSystem.ATTR_FOLDING_OPEN	オープン
	PhoneSystem.ATTR_FOLDING_CLOSE	クローズ
PhoneSystem.DEV_MAILBOX（メール状態）	PhoneSystem.ATTR_MAIL_NONE	メールなし
	PhoneSystem.ATTR_MAIL_RECEIVED	未読メールあり
	PhoneSystem.ATTR_MAIL_AT_CENTER	センターに未読メールあり
PhoneSystem.DEV_MESSAGEBOX（メッセージ状態）	PhoneSystem.ATTR_MESSAGE_NONE	メッセージなし
	PhoneSystem.ATTR_MESSAGE_RECEIVED	未読メッセージあり
	PhoneSystem.ATTR_MESSAGE_AT_CENTER	センターに未読メッセージあり
PhoneSystem.DEV_BATTERY（バッテリー状態）	PhoneSystem.ATTR_BATTERY_CHARGING	バッテリー充電中
	PhoneSystem.ATTR_BATTERY_FULL	バッテリーMAX
	PhoneSystem.ATTR_BATTERY_PARTIAL	バッテリーその他
PhoneSystem.DEV_SERVICEAREA（アンテナ状態）	PhoneSystem.ATTR_SERVICEAREA_OUTSIDE	アンテナ圏外
	PhoneSystem.ATTR_SERVICEAREA_INSIDE	アンテナ圏内
PhoneSystem.DEV_MANNER（マナーモード状態）	PhoneSystem.ATTR_MANNER_ON	マナーモードON
	PhoneSystem.ATTR_MANNER_OFF	マナーモードOFF
	PhoneSystem.ATTR_SIGNAL_OUTSIDE	圏外
PhoneSystem.DEV_MANNER（マナーモード状態）	PhoneSystem.ATTR_MANNER_ON	マナーモードON
	PhoneSystem.ATTR_MANNER_OFF	マナーモードOFF

▶ADFの設定

ADFは「GetUtn」と「GetSysInfo」を次のように設定してください。

GetUtn	terminalid,userid
GetSysInfo	yes

携帯電話のIDを取得する時には「getUtn」の「terminalid」、UIMカードのIDを取得する時には「userid」にチェックを入れる必要があります。端末情報を取得するには、「GetSysInfo」の「yes」にチェックを入れる必要があります。これら設定を行わないと、情報取得しようとした時に例外が発生します。

4-2 ネイティブ機能の呼び出し



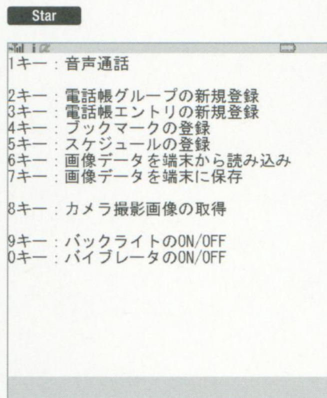
Star
http://book.mycom.co.jp/support/pc/star/42s.html



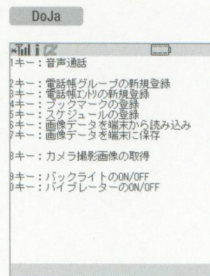
DoJa
http://book.mycom.co.jp/support/pc/star/d/42d.html

ネイティブ機能を呼び出すプログラムを作ります。ネイティブ機能とは携帯端末に最初から組み込まれている機能のことです。呼び出す機能は次の10種類です。

Star



DoJa



1キー	音声通話
2キー	電話帳グループの新規登録
3キー	電話帳エントリの新規登録
4キー	ブックマークの登録
5キー	スケジュールの登録
6キー	画像データを端末から読み込み
7キー	画像データを端末に保存
8キー	カメラ撮影画像の取得
9キー	バックライトのON/OFF
0キー	バイブレータのON/OFF

このプログラムは、次の2つのクラスで構成されています。

- **NativeExクラス（NativeEx.java）**
- **NativeCanvasクラス（NativeCanvas.java）**

プロジェクト名「**NativeEx**」でプロジェクトを作成してください。

▶NativeExクラス

NativeExクラスは、プログラムの本体となるクラスです。

Star NativeEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// ネイティブ機能の呼び出し（本体）
public class NativeEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {        // ⓒ
        NativeCanvas c = new NativeCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa NativeEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class NativeEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶NativeCanvasクラス

NativeCanvasクラスはキャンバスとなるクラスです。

Star

Star NativeCanvas.java

```
import com.docomostar.device.Camera;           // ⓐ
import com.docomostar.media.MediaImage;        //
import com.docomostar.media.MediaManager;      //
```

```
import com.docomostar.system.Bookmark;         //
import com.docomostar.system.PhoneBookGroup;   //
import com.docomostar.system.PhoneBook;        //
import com.docomostar.system.Phone;            //
import com.docomostar.system.ImageStore;       //
import com.docomostar.system.PhoneSystem;      //
import com.docomostar.system.Schedule;         //
import com.docomostar.system.ScheduleParam;    //
import com.docomostar.ui.Canvas;               //
import com.docomostar.ui.Dialog;               //
import com.docomostar.ui.Display;              //
import com.docomostar.ui.Graphics;             //
import com.docomostar.ui.Image;                //
import com.docomostar.util.ScheduleDate;       //
import java.util.Calendar;

// ネイティブ機能の呼び出し（キャンバス）
public class NativeCanvas extends Canvas
    implements Runnable {
    private int keyEvent = -999; // キーイベント
    private Camera camera;       // カメラ

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        // イメージ
        MediaImage mediaImage = null;
        Image      image      = null;

        // フラグ
        boolean backlightFlag = false;
        boolean vibratorFlag  = false;

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            if (image != null) g.drawImage(image,0,getHeight()-image.getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("1キー：音声通話",                0,24*1); // ⓑ
            g.drawString("2キー：電話帳グループの新規登録",    0,24*3); //
            g.drawString("3キー：電話帳エントリの新規登録",    0,24*4); //
            g.drawString("4キー：ブックマークの登録",         0,24*5); //
            g.drawString("5キー：スケジュールの登録",         0,24*6); //
            g.drawString("6キー：画像データを端末から読み込み", 0,24*7); //
            g.drawString("7キー：画像データを端末に保存",      0,24*8); //
```

Next page. ⬇

```
g.drawString("8 キー：カメラ撮影画像の取得 ",           0,24*10); //
g.drawString("9 キー：バックライトの ON/OFF",           0,24*12); //
g.drawString("0 キー：バイブレータの ON/OFF",           0,24*13); //
g.unlock(true);

// キーイベント
if (keyEvent==Display.KEY_1) {
    // 音声通話 ...❶
    Phone.call("117");
} else if (keyEvent==Display.KEY_2) {
    // 電話帳グループの新規登録 ...❷
    try {
        PhoneBookGroup.addEntry(" お友達 ");
    } catch (Exception e) {
    }
} else if (keyEvent==Display.KEY_3) {
    // 電話帳エントリの新規登録 ...❸
    try {
        PhoneBook.addEntry(
            " 伊佐坂うのみ ",
            " イササカウノミ ",
            new String[]{"01234567890"},
            new String[]{"unomi@npaka.net"},
            " お友達 ");
    } catch (Exception e) {
    }
} else if (keyEvent==Display.KEY_4) {
    // ブックマークの登録 ...❹
    try {
        Bookmark.addEntry(
            "http://npaka.net/",
            " ん・ぱか工房 ");
    } catch (Exception e) {
    }
} else if (keyEvent==Display.KEY_5) {
    // スケジュールの登録 ...❺
    try {                                             // ⓒ
        // スケジュール日時
        Calendar cal = Calendar.getInstance();
        cal.set(Calendar.YEAR,2009);
        cal.set(Calendar.MONTH,9-1);
        cal.set(Calendar.DATE,3);
        cal.set(Calendar.HOUR_OF_DAY,19);
        cal.set(Calendar.MINUTE,0);
        ScheduleDate date = new ScheduleDate(ScheduleDate.ONETIME,cal,null);

        // スケジュールパラメータ
        ScheduleParam param = new ScheduleParam();
        param.setDate(date);
```

```
                param.setDescription("バースディパーティー");
                param.setAlarm(true);

                // スケジュールの登録
                Schedule.addEntry(param);                       // ⓒ
            } catch (Exception e) {
            }
        } else if (keyEvent==Display.KEY_6) {
            // 画像データのメモリの解放
            if (image != null) {
                image.dispose();
                image      = null;
                mediaImage = null;
            }

            // 画像データを端末から読み込み …❻
            try {
                ImageStore is = ImageStore.selectEntry();
                if (is != null) {
                    mediaImage = is.getImage();
                    mediaImage.use();
                    image = mediaImage.getImage();
                }
            } catch (Exception e) {
            }
        } else if (keyEvent==Display.KEY_7) {
            // 画像データを端末に保存 …❼
            try {
                if (mediaImage != null) {
                    ImageStore.addEntry(mediaImage);
                } else {
                    Dialog dialog = new Dialog(Dialog.DIALOG_INFO,"情報");// ⓓ
                    dialog.setText("「画像データを端末から読み込み」か"+
                        "「カメラ撮影画像の取得」で"+
                        "画像データを取得してから選択してください。");
                    dialog.show();
                }
            } catch (Exception e) {
            }
        } else if (keyEvent==Display.KEY_8) {
            // 画像データのメモリの解放
            if (image != null) {
                image.dispose();
                image      = null;
                mediaImage = null;
            }

            // カメラ撮影画像の取得 …❽
            try {
```

Next page. ⬇

```
                    if (camera==null) {
                        camera = Camera.getCamera(0);
                        camera.setImageSize(120,120);
                        camera.setAttribute(Camera.DEV_QUALITY,Camera.ATTR_QUALITY_LOW);
                    }
                    camera.takePicture();
                    if (camera.getNumberOfImages() != 0) {
                        mediaImage = (MediaImage)camera.getImage(0); // ⓔ
                        mediaImage.use();
                        image = mediaImage.getImage();
                    }
                } catch (Exception e) {
                }
            } else if (keyEvent==Display.KEY_9) {
                // バックライトの ON/OFF ...❾
                backlightFlag = !backlightFlag;
                if (backlightFlag) {
                    PhoneSystem.setAttribute(
                        PhoneSystem.DEV_BACKLIGHT,
                        PhoneSystem.ATTR_BACKLIGHT_ON);
                } else {
                    PhoneSystem.setAttribute(
                        PhoneSystem.DEV_BACKLIGHT,
                        PhoneSystem.ATTR_BACKLIGHT_OFF);
                }
            } else if (keyEvent==Display.KEY_0) {
                // バイブレータの ON/OFF ...❾
                vibratorFlag = !vibratorFlag;
                if (vibratorFlag) {
                    PhoneSystem.setAttribute(
                        PhoneSystem.DEV_VIBRATOR,
                        PhoneSystem.ATTR_VIBRATOR_ON);
                } else {
                    PhoneSystem.setAttribute(
                        PhoneSystem.DEV_VIBRATOR,
                        PhoneSystem.ATTR_VIBRATOR_OFF);
                }
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // キーイベントの処理
```

```
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa NativeCanvas.java

ⓐ

```
import com.nttdocomo.device.Camera;
import com.nttdocomo.system.Bookmark;
import com.nttdocomo.system.ImageStore;
import com.nttdocomo.system.PhoneBook;
import com.nttdocomo.system.PhoneBookGroup;
import com.nttdocomo.system.Schedule;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.PhoneSystem;
import com.nttdocomo.util.Phone;
import com.nttdocomo.util.ScheduleDate;
import java.util.Calendar;
```

ⓑ

```
            g.drawString("1キー：音声通話",             0,12*1);
            g.drawString("2キー：電話帳グループの新規登録",  0,12*3);
            g.drawString("3キー：電話帳エントリの新規登録",  0,12*4);
            g.drawString("4キー：ブックマークの登録",       0,12*5);
            g.drawString("5キー：スケジュールの登録",       0,12*6);
            g.drawString("6キー：画像データを端末から読み込み",0,12*7);
            g.drawString("7キー：画像データを端末に保存",    0,12*8);
            g.drawString("8キー：カメラ撮影画像の取得",      0,12*10);
            g.drawString("9キー：バックライトのON/OFF",     0,12*12);
            g.drawString("0キー：バイブレータのON/OFF",     0,12*13);
```

ⓒ

```
                ScheduleDate date = new ScheduleDate(ScheduleDate.ONETIME);
                date.set(Calendar.YEAR,2009);
                date.set(Calendar.MONTH,9-1);
```

Next page. ⬇

```
                    date.set(Calendar.DATE,3);
                    date.set(Calendar.HOUR_OF_DAY,19);
                    date.set(Calendar.MINUTE,0);
                    Schedule.addEntry("バースディパーティー",date,true);
```

ⓓ

```
                        dialog.setFont(Font.getFont(Font.SIZE_SMALL));
```

ⓔ

```
                        mediaImage = camera.getImage(0);
```

NativeCanvasクラス …❶：音声通話

iアプリから電話をかけるには、Phoneクラスのcall()メソッドを使います。

［Phoneクラス］

static void call(*phoneNumber*)

［解説］ 音声通話

［引数］ *phoneNumber*　電話番号：String型

引数の*phoneNumber*（電話番号）には電話番号を表す文字列を指定します。 数字（0～9）と # と ＊ を使います。 - は無視されます。

NativeCanvasクラス …❷：電話帳グループの新規登録

iアプリからネイティブの電話帳に電話帳グループを新規登録するには、PhoneBookGroupクラスのaddEntry()メソッドを使います。

［PhoneBookGroupクラス］

static void addEntry(*name*)

［解説］ 電話帳グループの新規登録

［引数］ *name*　グループ名：String型

［戻り値］ グループID（キャンセルした時は-1）

NativeCanvasクラス …❸：電話帳エントリの新規登録

iアプリからネイティブの電話帳に電話帳エントリを新規登録するには、PhoneBookクラスのaddEntry()メソッドを使います。

［PhoneBookクラス］

static int[] addEntry(*name, kana, phoneNumbers, mailAddress, groupId*)

［解説］ 電話帳エントリの新規登録
［引数］ *name* 名前：String型
kana ヨミガナ：String型
phoneNumbers 電話番号：String[]型
mailAddress メールアドレス：String[]型
groupID グループID（指定しない時は-1）：int型
［戻り値］ エントリIDとグループID（キャンセルした時はnull）

NativeCanvasクラス …❹：ブックマークの登録

iアプリからブックマークを登録するには、Bookmarkクラスのaddentry()メソッドを使います。

［Bookmarkクラス］

static void addEntry(*url, title*)

［解説］ ブックマークの登録
［引数］ *url* URL：String型
title タイトル：String型

NativeCanvasクラス …❺：スケジュールの登録

ネイティブのスケジューラにスケジュールを登録するには、ScheduleクラスのaddEntry()メソッドを使います。

Star ［Scheduleクラス］

static boolean addEntry(*param*)

［解説］ スケジュールの登録
［引数］ *param* スケジュールパラメータ：ScheduleParam型

ScheduleParamクラスは、スケジュール情報を保持するクラスです。setDate()メソッドで日付、setDescription()メソッドで説明、setAlarm()メソッドでアラームの有無を指定します。

Star ［ScheduleParamクラス］

void setDate(*date*)

［解説］ 日付の指定
［引数］ *date* 日付：Date型

Star ［ScheduleParamクラス］

void setDescription(*description*)

［解説］ 説明の指定
［引数］ *description* 日付：Description型

Star ［ScheduleParamクラス］

void setAlarm(*flag*)

［解説］ アラームの有無の指定
［引数］ *flag* アラームの有無：boolean型

ScheduleDateクラスは、スケジュール日時を保持するクラスです。コンストラクタを2つ持っています。

Star ［ScheduleDateクラス］

ScheduleDate(*type*, *start*, *end*)

［解説］ ScheduleDateクラスのコンストラクタ
［引数］ *type* 日時種別：int型
start 開始日：Calendar型
end 終了日：Calendar型

Star ［ScheduleDateクラス］

ScheduleDate(*type, repeatCount, spanSet, start, end***)**

［解説］ ScheduleDateクラスのコンストラクタ
［引数］ *type* 日時種別：int型
repeatCount 繰り返し回数：int型
spanSet 期間の集合：int[]型
start 開始日：Calendar型
end 終了日：Calendar型

引数*type*（日時種別）は次の定数を指定します。

ScheduleDate.ONETIME	1回限り
ScheduleDate.DAILY	毎日
ScheduleDate.WEEKLY	毎週
ScheduleDate.MONTHLY	毎月
ScheduleDate.YEARLY	毎年

引数*spanSet*（期間の集合）は月や曜日を指定したい時に使います。

DoJa
ネイティブのスケジューラにスケジュールを登録するには、ScheduleクラスのaddEntry()メソッドを使います。

DoJa ［Scheduleクラス］

static boolean addEntry(*description, date, alarm***)**

［解説］ スケジュールの登録
［引数］ *description* 内容：String型
date 日時：ScheduleDate型
alarm アラームを鳴らすか：boolean型
［戻り値］ 登録が成功したか

ScheduleDateクラスは、スケジュール日時を保持するクラスです。コンストラクタを1つ持っています。

DoJa ［ScheduleDateクラス］

ScheduleDate(*type*)

［解説］ ScheduleDateクラスのコンストラクタ
［引数］ *type* 日時種別：int型

引数の*type*（日時種別）には次の定数を指定します。

ScheduleDate.ONETIME	1回限り
ScheduleDate.DAILY	毎日
ScheduleDate.WEEKLY	毎週
ScheduleDate.MONTHLY	毎月
ScheduleDate.YEARLY	毎年

「１回限り」以外は使えるかどうかは機種依存となります。
ScheduleDateオブジェクトに日時を指定するには、set()メソッドを使います。

［ScheduleDateクラス］

void set(*attr, value*)

［解説］ 属性値の指定
［引数］ *attr* 属性：int型
value 値：int型

引数の*attr*（属性）には次の定数を指定します。

Calendar.YEAR	年
Calendar.MONTH	月 - 1
Calendar.DAY_OF_MONTH	日
Calendar.DAY_OF_WEEK	曜日
Calendar.HOUR_OF_DAY	時
Calendar.MINUTE	分

引数の*value*（値）には、属性に応じた値を指定します。属性がMONTHの時は［月 - 1］の値（1月は0）、属性がDAY_OF_WEEKの時は次の定数を指定します。

Calendar.MONDAY	月
Calendar.TUESDAY	火
Calendar.WEDNESDAY	水
Calendar.THURSDAY	木
Calendar.FRIDAY	金
Calendar.SATURDAY	土
Calendar.SUNDAY	日

NativeCanvasクラス ...❻：画像データを端末から読み込み

画像データを端末のデータフォルダから読み込むには、ImageStoreクラスのselectEntry()メソッドを使います。サイズが大きすぎる写真や、再配布不可識別子の入った画像データは読み込めません。

［ImageStoreクラス］

static ImageStore selectEntry()

［解説］　画像データを端末のデータフォルダから選択
［戻り値］　ImageStoreオブジェクト（キャンセルした時はnull）

ImageStoreオブジェクトのgetImage()メソッドでMediaImageオブジェクト、getInputStream()メソッドで入力ストリームを取得することができます。

［ImageStoreクラス］

ImageStore getImage()

［解説］　イメージの読み込み
［戻り値］　MediaImageイメージ

［ImageStoreクラス］

InputStream getInputStream()

［解説］　イメージの読み込み
［戻り値］　入力ストリーム

NativeCanvasクラス ...❼：画像データを端末に保存

画像データを端末のデータフォルダに保存するには、ImageStoreクラスのaddEntry()メソッドを使います。

［ImageStoreクラス］

static int addEntry(*mediaImage*)

［解説］　画像データを端末に保存
［引数］　*mediaImage*　メディアイメージ：MediaImage型
［戻り値］　エントリID（キャンセルした時は-1）

NativeCanvasクラス ...❽：カメラ撮影画像の取得

カメラを制御するにはCameraクラスのgetCamera()メソッドでCameraクラスのオブジェクトを取得します。

［Cameraクラス］

static Camera getCamera(*id*)

［解説］ Cameraオブジェクトの取得
［引数］ *id* カメラID：int型
［戻り値］ Cameraオブジェクト

引数の*id*（カメラID）は、複数カメラを持つ端末でどのカメラを制御するかを指定するもので、0から順番に割り振られています。

次に、CameraクラスのsetImageSize()メソッドで、撮影する写真の画像サイズを指定します。

［Cameraクラス］

void setImageSize(*width*, *height*)

［解説］ 写真の画像サイズの指定
［引数］ *width* 幅 ：int型
height 高さ：int型

実際にどの画像サイズで撮れるかは機種依存で、ここで指定した値に近いものが選択されます。

次に、CameraクラスのsetAttribute()メソッドで、連写モード・画質・マイク音量を指定します。

［Cameraクラス］

void setAttribute(*attr*, *value*)

［解説］ 属性の指定
［引数］ *attr* 属性：int型
value 値 ：int型

attr（属性）と*value*（値）には次の定数を指定します。

属性	値	意味
DEV_CONTINUOUS_SHOT（連写モード）	ATTR_CONTINUOUS_SHOT_OFF	OFF
	ATTR_CONTINUOUS_SHOT_ON	ON
DEV_QUALITY（画質）	ATTR_QUALITY_HIG	高
	ATTR_QUALITY_STANDARD	標準
	ATTR_QUALITY_LOW	低
DEV_SOUND（マイク音量）	ATTR_VOLUME_MIN	最小0
	ATTR_VOLUME_MAX	最大127

Cameraクラスのtake Picture()メソッドを呼ぶと、カメラが起動します。そして、写真を撮るかキャンセルを行うと、iアプリに制御が戻ります。写真を撮ったかどうかは、getNumberOfImage()メソッドの戻り値が0かどうかを判定します。

［Cameraクラス］

int getNumberOfImage()

［解説］ Cameraオブジェクトが保持している撮影画像数の取得
［戻り値］ Cameraオブジェクトが保持している撮影画像数

Cameraオブジェクトのget Image()メソッドでImageオブジェクト、getInputStream()メソッドで入力ストリームを取得することができます。

［Cameraクラス］

Image getImage(*index*)

［解説］ イメージの取得
［引数］ *index* 写真のインデックス：int型
［戻り値］ Imageオブジェクト

［Cameraクラス］

InputStream getInputStream(*index*)

［解説］ イメージの取得
［引数］ *index* 写真のインデックス：int型
［戻り値］ 入力ストリーム

引数の*index*（写真のインデックス）は連写モードで撮影した時、何枚目の写真を取得するかを指定するものです。連写モードを使ってない時は0を指定します。

Cameraオブジェクトが保持している撮影画像は、次にtakePicture()メソッドかdisposeImages()メソッドを呼んだ時に破棄されます。

［Cameraクラス］

void disposeImages()

［解説］ 撮影画像の破棄

NativeCanvasクラス ...❾：バックライトとバイブレータのON/OFF

バックライトとバイブレータを制御するには、PhoneSystemクラスのsetAttribute()メソッドを使います。

［PhoneSystemクラス］

static void setAttribute(*attr, value*)

［解説］ 属性の指定

［引数］ *attr* 属性：int型

value 値：int型

引数の*attr*（属性）と*value*（値）には次の定数を指定します。

属性	値	意味
PhoneSystem.DEV_BACKLIGHT（バックライト）	PhoneSystem.ATTR_BACKLIGHT_ON	ON
	PhoneSystem.ATTR_BACKLIGHT_OFF	OFF
PhoneSystem.DEV_VIBRATOR（バイブレータ）	PhoneSystem.ATTR_VIBRATOR_ON	ON
	PhoneSystem.ATTR_VIBRATOR_OFF	OFF

▶ADFの設定

ADFは「UseTelephone」と「AccessUserInfo」を次のように設定してください。

UseTelephone	call
AccessUserInfo	yes

音声通話を行う時は「UseTelephone」の「call」にチェックを入れる必要があります。電話帳やブックマークなどのユーザ情報にアクセスする時は「AccessUserInfo」の「yes」にチェックを入れる必要があります。これら設定を行わないと、ネイティブ機能を呼び出した時に例外が発生します。

4-3 コードリーダの利用



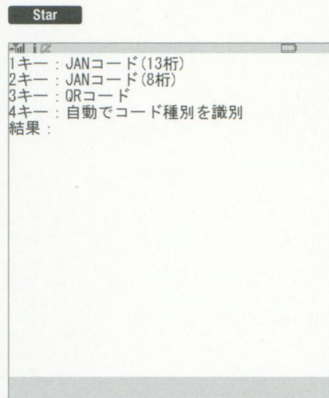
Star
http://book.mycom.co.jp/support/pc/star/43s.html



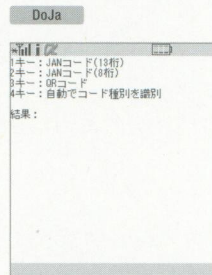
DoJa
http://book.mycom.co.jp/support/pc/star/d/43d.html

コードリーダを使用してJANコードとQRコードの情報を読み取るプログラムを作ります。

Star



DoJa



1キー	JANコード（13桁）
2キー	JANコード（8桁）
3キー	QRコード
4キー	自動でコード種別を識別

JANコードは、世界共通コードとしてお店に流通しているほとんどの商品に付いているバーコードです。国コード、メーカーコード、商品コード等の情報が含まれています。また、JANコードには13桁の標準タイプと8桁の短縮タイプがあり、通常は標準タイプを使い、印刷面積が足りない時だけ短縮タイプを使います。

QRコードは、数cm四方のスペースに数百文字の情報を記録できる2次元コードです。携帯電話では主にサイトのURLや個人のアドレスを広告や名刺から取り込む時に使われています。





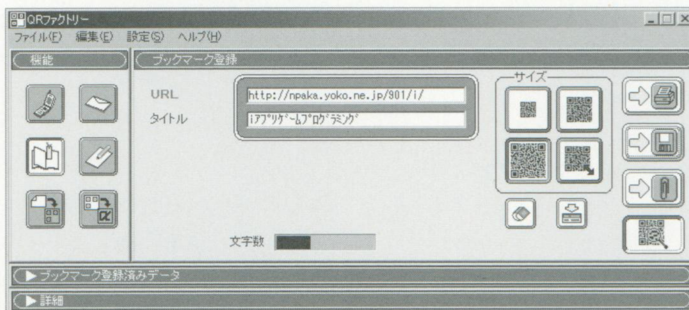
このプログラムは、次の2つのクラスで構成されています。

- **CodeReaderExクラス（CodeReaderEx.java）**
- **CodeReaderCanvasクラス（CodeReaderCanvas.java）**

プロジェクト名「**CodeReaderEx**」でプロジェクトを作成してください。

Column» QRコードの作成

QRコードを作成するには、NTTドコモが無償で提供しているツール**QRファクトリー**を使います。以下のサイトからダウンロードして、インストールしてください。QRコードの中に含ませる情報を入力した後、画面右の作成ボタンでQRコードの画像ファイルを作成できます。



- QRファクトリー
  http://www.nttdocomo.co.jp/service/imode/make/content/barcode/tool/

▶CodeReaderExクラス

CodeReaderExクラスは、プログラムの本体となるクラスです。

Star CodeReaderEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// コードリーダーの利用 ( 本体 )
public class CodeReaderEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {           // ⓒ
        CodeReaderCanvas c = new CodeReaderCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa CodeReaderEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class CodeReaderEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶CodeReaderCanvasクラス

CodeReaderCanvasクラスは、キャンバスとなるクラスです。

Star CodeReaderCanvas.java

```
import com.docomostar.device.CodeReader; // ⓐ
import com.docomostar.ui.Canvas;         //
import com.docomostar.ui.Dialog;         //
import com.docomostar.ui.Display;        //
import com.docomostar.ui.Graphics;       //
```

Next page.

```
// コードリーダの利用 ( キャンバス )
public class CodeReaderCanvas extends Canvas
    implements Runnable {
    private int keyEvent = -999; // キーイベント

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        // 情報
        String info = "";

        // コードリーダオブジェクトの取得 …❶
        CodeReader codeReader = CodeReader.getCodeReader(0);
        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("1 キー：JAN コード (13 桁 )",    0,24*1);  // ⓑ
            g.drawString("2 キー：JAN コード (8 桁 )",     0,24*2);  //
            g.drawString("3 キー：QR コード ",              0,24*3);  //
            g.drawString("4 キー：自動でコード種別を識別 ", 0,24*4);  //
            g.drawString(" 結果：" +info,0,24*5);                     //
            g.unlock(true);                                          //

            // キーイベント
            if (keyEvent==Display.KEY_1) {
                // バーコード (JAN13)
                try {
                    codeReader.setCode(CodeReader.CODE_JAN13);  // …❷
                    codeReader.read();                          // …❸
                    int type = codeReader.getResultType();      // …❹
                    if (type==CodeReader.TYPE_ASCII  ||
                        type==CodeReader.TYPE_NUMBER ||
                        type==CodeReader.TYPE_STRING) {
                        info = codeReader.getString();          // …❺
                    } else {
                        info = "";
                    }
                } catch (Exception e) {
                    showDialog(" 読み取り失敗 ");
                }
            } else if (keyEvent==Display.KEY_2) {
                // バーコード (JAN8)
                try {
                    codeReader.setCode(CodeReader.CODE_JAN8);
```

```
                codeReader.read();
                int type = codeReader.getResultType();
                if (type==CodeReader.TYPE_ASCII  ||
                    type==CodeReader.TYPE_NUMBER ||
                    type==CodeReader.TYPE_STRING) {
                    info = codeReader.getString();
                } else {
                    info = "";
                }
            } catch (Exception e) {
                showDialog("読み取り失敗");
            }
        } else if (keyEvent==Display.KEY_3) {
            // QRコード種別
            try {
                codeReader.setCode(CodeReader.CODE_QR);
                codeReader.read();
                int type = codeReader.getResultType();
                if (type==CodeReader.TYPE_ASCII  ||
                    type==CodeReader.TYPE_NUMBER ||
                    type==CodeReader.TYPE_STRING) {
                    info = codeReader.getString();
                } else {
                    info = "";
                }
            } catch (Exception e) {
            }
        } else if (keyEvent==Display.KEY_4) {
            // 自動でコード種別を識別
            try {
                codeReader.setCode(CodeReader.CODE_AUTO);
                codeReader.read();
                int type = codeReader.getResultType();
                if (type==CodeReader.TYPE_ASCII  ||
                    type==CodeReader.TYPE_NUMBER ||
                    type==CodeReader.TYPE_STRING) {
                    info = codeReader.getString();
                } else {
                    info = "";
                }
            } catch (Exception e) {
                showDialog("読み取り失敗");
            }
        }
        keyEvent = -999;

        // スリープ
        try {
            Thread.sleep(100);
```

Next page.

chapter

1

2

3

4

5

6

7

8

a

```
            } catch (Exception e) {
            }
        }
    }

    // ダイアログの表示                    // ⓒ
    void showDialog(String text) {  //
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー");
        dialog.setText(text);
        dialog.show();
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa CodeReaderCanvas.java

ⓐ

```
import com.nttdocomo.device.CodeReader;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;
```

ⓑ

```
            g.drawString("1キー：JANコード(13桁)",      0,12);
            g.drawString("2キー：JANコード(8桁)",       0,24);
            g.drawString("3キー：QRコード",             0,36);
            g.drawString("4キー：自動でコード種別を識別", 0,48);
            g.drawString("結果："+info,0,72);
            g.unlock(true);
```

ⓒ

```
    // ダイアログの表示
    private void showDialog(String text) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー");
        dialog.setFont(Font.getFont(Font.SIZE_SMALL));
```

CodeReaderCanvasクラス ...❶：コードリーダオブジェクトの取得

コードリーダオブジェクトを取得するには、CodeReaderクラスのgetCodeReader()メソッドを使います。

［CodeReaderクラス］

static CodeReader getCodeReader(*id*)

［解説］ コードリーダオブジェクトの取得

［引数］ *id* カメラID：int型

［戻り値］ CodeReaderオブジェクト

引数の*id*（カメラID）は、複数カメラを持つ端末でどのカメラを制御するかを指定するものです。0から順番に割り振られています。

CodeReaderCanvasクラス ...❷：コード種別の指定

読み取るコード種別を指定するには、CodeReaderクラスのsetCode()メソッドを使います。

［CodeReaderクラス］

void setCode(*type*)

［解説］ 読み取るコード種別の指定

［引数］ *type* コード種別：int型

引数*type*（コード種別）には次の定数を指定します。

CodeReader.CODE_JAN13	JANコード（13桁）
CodeReader.CODE_JAN8	JANコード（8桁）
CodeReader.CODE_QR	QRコード
CodeReader.CODE_OCR	文字認識
CodeReader.CODE_AUTO	自動でコード種別を識別

QRコードとJANコードはほとんどの端末で対応していますが、文字認識が可能な端末は限られます。

CodeReaderCanvasクラス ...❸：コードの読み取り

コードの読み取りを行うには、CodeReaderクラスのread()メソッドを使います。

［CodeReaderクラス］

void read()

［解説］ コードの読み取り

CodeReaderCanvasクラス ...❹：コード読み取り結果種別の取得

コード読み取り結果種別を取得するには、CodeReaderクラスのgetResultType()メソッドを使います。

［CodeReaderクラス］

int getResultType()

［解説］ コードの読み取り結果種別の取得
［戻り値］ コード読み取り結果種別

戻り値として、次の定数がコード読み取り結果種別として返されます。

CodeReader.TYPE_ASCII	ASCII文字列
CodeReader.TYPE_BINARY	バイナリ
CodeReader.TYPE_NUMBER	数字
CodeReader.TYPE_STRING	文字列
CodeReader.TYPE_UNKNOWN	型が不明

CodeReaderCanvasクラス ...❺：コード認識結果の取得

コード認識結果を文字列として取得するには、CodeReaderクラスのgetString()メソッドを使います。

［CodeReaderクラス］

String getString()

［解説］ コード認識結果の取得
［戻り値］ コード認識結果

バイトデータとして取得するには、getByte()メソッドを使います。

［CodeReaderクラス］

byte[] getByte()

［解説］ コード認識結果の取得
［戻り値］ コード認識結果

4-4 文字入力の処理

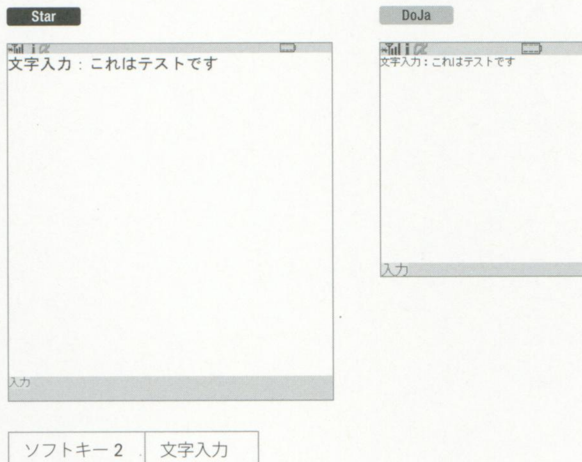


Star
http://book.mycom.co.jp/support/pc/star/44s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/44d.html

文字入力を行うプログラムを作ります。ソフトキー 1を押すことで、文字入力機能IME（Input Method Editor）を呼び出します。そこで文字入力を行い確定すると、キャンバスに入力した文字列が表示されます。



ソフトキー 2	文字入力

このプログラムは、次の2つのクラスで構成されています。

- **IMEExクラス（IMEEx.java）**
- **IMECanvasクラス（IMECanvas.java）**

プロジェクト名「**IMEEx**」でプロジェクトを作成してください。

▶IMEExクラス

IMEExクラスは、プログラムの本体となるクラスです。

Star IMEEx.java

```
import com.docomostar.StarApplication; // ⓐ
```

Next page. ↓

chapter
1
2
3
4
5
6
7
8
a

```
import com.docomostar.ui.Display;       //

// 文字入力を処理する(本体)
public class IMEEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {     // ⓒ
        IMECanvas c = new IMECanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa IMEEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class IMEEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶IMECanvasクラス

IMECanvasクラスはキャンバスとなるクラスです。

Star IMECanvas.java

```
import com.docomostar.ui.Canvas;   // ⓐ
import com.docomostar.ui.Display;  //
import com.docomostar.ui.Graphics; //
import com.docomostar.ui.TextBox;  //

// 文字入力を処理する（キャンバス）
public class IMECanvas extends Canvas
    implements Runnable {
    private int    keyEvent = -999; // キーイベント
    private String info     = "";   // 情報

    // 処理
    public void run() {
        // グラフィックス
        Graphics g=getGraphics();

        // ソフトラベル
        setSoftLabel(SOFT_KEY_1,"入力");

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("文字入力："+info,0,24);  // ⓑ
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_SOFT1) {
                // IMEの呼び出し …❶
                imeOn(info,TextBox.DISPLAY_ANY,TextBox.KANA);
            }
            keyEvent=-999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }
```

Next page. ⬇

```
        // IME イベントの処理 …❷
        public void processIMEEvent(int type,String text) {
            // 入力確定
            if (type==IME_COMMITTED) info = text;
        }

        // キーイベントの処理
        public void processEvent(int type, int param) {
            if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
        }

        // 描画
        public void paint(Graphics g) {}
    }
```

DoJa IMECanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.TextBox;
```

ⓑ

```
                g.drawString("文字入力："+info,0,12);
```

IMECanvasクラス ...❶：IMEの呼び出し

IMEを呼び出すには、Canvasクラスのime0n()メソッドを使います。

［Canvasクラス］

void imeOn(*text, displayMode, inputMode***)**

［解説］ IMEの呼び出し

［引数］ *text* 初期文字列：String型

displayMode 文字列表示モード：int型

inputMode 初期入力モード：int型

引数の*displayMode*（文字列表示モード）には次の定数を指定します。

TextBox.DISPLAY_ANY	表示文字列をそのまま表示
TextBox.DISPLAY_PASSWORD	表示文字列を隠蔽（パスワード）

引数の*inputMode*（初期入力モード）には次の定数を指定します。

TextBox.NUMBER	数字入力
TextBox.ALPHA	アルファベット入力
TextBox.KANA	かな漢字入力

IMECanvasクラス ...❷：IMEイベントの処理

IMEで入力した文字は、CanvasクラスのprocessIMEEvent()メソッドに渡されます。これをオーバーライドすることによって、入力文字を取得することができます。

［Canvasクラス］

void processIMEEvent(*type, text***)**

［解説］ IMEイベント発生時に呼ばれる

［引数］ *type* イベント種別：int型

text 入力した文字列：String型

引数の*type*（イベント種別）には次の定数が渡されます。

Canvas.IME_COMMITTED	入力が確定
Canvas.IME_CANCELED	入力が取り消し

chapter
1
2
3
4
5
6
7
8
a

4-5 iアプリからのブラウザ起動とiアプリ起動



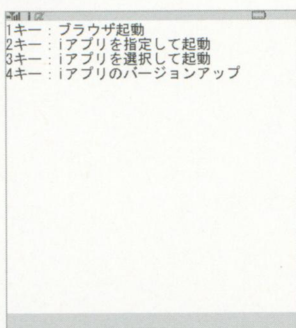
Star
http://book.mycom.co.jp/support/pc/star/45s.html



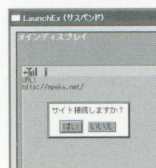
DoJa
http://book.mycom.co.jp/support/pc/star/d/45d.html

iアプリから**ブラウザ起動**や**iアプリ起動**を行うプログラムを作ります。

Star



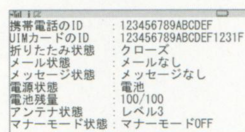
1キー →



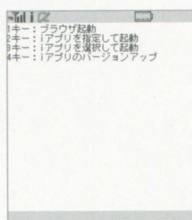
→



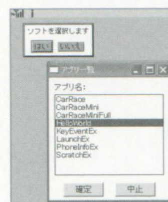
2キー →



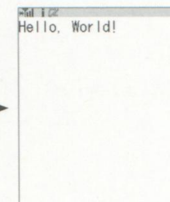
DoJa



3キー →



→



このプログラムは、次の2つのクラスで構成されています。

- **LaunchExクラス（LaunchEx.java）**
- **LaunchCanvasクラス（LaunchCanvas.java）**

プロジェクト名「**LaunchEx**」でプロジェクトを作成してください。

▶LaunchExクラス

LaunchExクラスは、プログラムの本体となるクラスです。

Star LaunchEx.java

```
import com.docomostar.StarApplication;
import com.docomostar.ui.Display;

// iアプリからのブラウザ起動とiアプリ起動（本体）
public class LaunchEx extends StarApplication {

    // iアプリの開始
    public void started(int launchType) {
        LaunchCanvas c = new LaunchCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa LaunchEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class LaunchEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶LaunchCanvasクラス

LaunchCanvasクラスはキャンバスとなるクラスです。

Star LaunchCanvas.java

```
import com.docomostar.StarApplication;               // ⓐ
import com.docomostar.StarApplicationManager;        //
import com.docomostar.system.ApplicationStore;       //
import com.docomostar.system.Launcher;               //
import com.docomostar.ui.Canvas;                     //
import com.docomostar.ui.Dialog;                     //
import com.docomostar.ui.Display;                    //
import com.docomostar.ui.Graphics;                   //

// iアプリからのブラウザ起動とiアプリ起動（キャンバス）
```

Next page.

```
public class LaunchCanvas extends Canvas
    implements Runnable {
    private int keyEvent = -999; // キーイベント

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("1 キー：ブラウザ起動 ",              0,24*1); // ⓑ
            g.drawString("2 キー：i アプリを指定して起動 ",     0,24*2); //
            g.drawString("3 キー：i アプリを選択して起動 ",     0,24*3); //
            g.drawString("4 キー：i アプリのバージョンアップ ", 0,24*4); //
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_1) {
                // ブラウザ起動 …❶
                Launcher.launch(Launcher.LAUNCH_BROWSER, // ⓒ
                    new String[]{"http:// npaka.net/"});
            } else if (keyEvent==Display.KEY_2) {
                // i アプリを指定して起動 …❷
                try {
                    Launcher.launch(Launcher.LAUNCH_STARAPPLI,  // ⓓ
                        new String[]{
                        StarApplicationManager.getSourceURL()+  // ⓔ
                        "PhoneInfoEx.jam"});
                } catch (Exception e) {
                    Dialog dialog = new Dialog(Dialog.DIALOG_INFO," 情報 ");
                                                                // ⓕ
                    dialog.setText("i アプリ起動に失敗しました ");
                    dialog.show();
                }
            } else if (keyEvent==Display.KEY_3) {
                // i アプリを選択して起動 …❸
                ApplicationStore as = null;
                try {
                    as = ApplicationStore.selectEntry();
                } catch (Exception e) {
                }
                if (as  !=  null) {
                    Launcher.launch(Launcher.LAUNCH_AS_LAUNCHER,  // ⓖ
                        new String[]{""+as.getId()});
```

```
                }
            } else if (keyEvent==Display.KEY_4) {
                // iアプリのバージョンアップ ...❹
                StarApplication.getThisStarApplication().   // ⓗ
                    getStarApplicationManager().upgrade();  //
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```


DoJa LaunchCanvas.java

ⓐ

```
import com.nttdocomo.system.ApplicationStore;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.IApplication;
```

ⓑ

```
            g.drawString("1キー：ブラウザ起動",           0,12*1);
            g.drawString("2キー：iアプリを指定して起動",    0,12*2);
            g.drawString("3キー：iアプリを選択して起動",    0,12*3);
            g.drawString("4キー：iアプリのバージョンアップ",0,12*4);
```

ⓒ

```
                IApplication.getCurrentApp().launch(
                    IApplication.LAUNCH_BROWSER,
```

chapter

1

2

3

4

5

6

7

8

a

ⓓ

```
                IApplication.getCurrentApp().launch(
                    IApplication.LAUNCH_IAPPLI,
```

ⓔ

```
                    IApplication.getCurrentApp().getSourceURL()+
```

ⓕ

```
                IApplication.getCurrentApp().launch(
                    IApplication.LAUNCH_AS_LAUNCHER,
```

ⓖ

```
            IApplication.getCurrentApp().launch(
                IApplication.LAUNCH_VERSIONUP,
                null);
```

LaunchCanvas ...❶：ブラウザ起動

Star

iアプリからブラウザを起動するには、Launcherクラスのlaunch()メソッドを使います。

Star ［Launcherクラス］

String[] launch(*target, args***)**

［解説］ ブラウザ起動

［引数］ *target* 起動するアプリケーションの種類：int型

args 起動するアプリケーションに渡す引数：String[]型

第1引数*target*にLauncher.LAUNCH_BROWSER、第2引数*args*にURLを指定します。

DoJa

iアプリからブラウザを起動するには、IApplicationクラスのlaunch()メソッドを使います。

DoJa ［IApplicationクラス］

String[] launch(*target, args***)**

［解説］ ブラウザ起動

［引数］ *target* 起動するアプリケーションの種類：int型

args 起動するアプリケーションに渡す引数：String[]型

第1引数*target*にIApplication.LAUNCH_BROWSER、第2引数*args*にURLを指定します。

Column»　iアプリからiアプリ起動時のパラメータ渡し

iアプリからiアプリにパラメータを渡したい時は、第2引数の配列の2つ目の要素以降に、属性と値を順に指定します。パラメータは最大16個まで指定でき、属性名と値の合計バイト数が最大255バイトまでという制限があります。

```
new String[]{"ADFの絶対パス", "パラメータの属性名1", "パラメータの値1",
            "パラメータの属性名2", "パラメータの値2", …}
```

iアプリで渡されたパラメータを取得するには、StarはStarApplicationManagerクラス、DoJaはIApplicationクラスのgetParameter()メソッドを使います。

LaunchCanvas ...❷：iアプリを指定して起動

Star

iアプリからiアプリを指定して起動する時も同様に、Launcherクラスのlaunch()メソッドを使います。第1引数にLauncher.LAUNCH_STARAPPLI、第2引数に起動するiアプリのADFの絶対パスを指定します。

iアプリが端末内に存在しない時は、例外が発生します。また、起動するiアプリと起動されるiアプリのダウンロード元ホストが同じでなければなりません。起動中のiアプリ自身を起動することもできません。

ダウンロード元URLを取得するには、StarApplicationManagerクラスのgetSourceURL()メソッドを使います。

Star［StarApplicationManagerクラス］

String getSourceURL()

［解説］　ダウンロード元URLの取得
［戻り値］　ダウンロード元URL

ADFの絶対パスが「http://npaka.net/star10/doja_dl/LaunchEx.jam」の時、「http://npaka.net/star10/doja_dl/」が返ります。

DoJa

iアプリからiアプリを指定して起動する時も同様に、IApplicationクラスのlaunch()メソッドを使います。第1引数にIApplication.LAUNCH_IAPPLI、第2引数に起動するiアプリのADFの絶対パスを指定します。

ダウンロード元URLを取得するには、IApplicationクラスのgetSourceURL()メソッドを使います。

DoJa ［IApplicationクラス］

String getSourceURL()

［解説］　ダウンロード元URLの取得
［戻り値］　ダウンロード元URL

Star/DoJa

エミュレータ上のiアプリのダウンロードURLを設定するには、iαppliToolのメニューの［設定］-［アプリケーション動作環境設定］-［ネットワーク設定］を選択し、「ADFのURL」に指定します。今回は「http://npaka.net/star10/doja_dl/」と指定します。

LaunchCanvas ...❸：iアプリを選択して起動

Star

iアプリから端末内のiアプリを選択して起動する時も同様に、Launcherクラスのlaunch()メソッドを使います。第1引数にLauncher.LAUNCH_AS_LAUNCHER、第2引数にエントリIDを指定します。

DoJa

iアプリから端末内のiアプリを選択して起動する時も同様に、IApplicationクラスのlaunch()メソッドを使います。第1引数にIApplication.LAUNCH_AS_LAUNCHER、第2引数にエントリIDを指定します。

Star/DoJa

エントリIDは端末内のiアプリを一意に示すIDで、ApplicationStoreクラスで取得できます。selectEntry()メソッドでApplicationStoreオブジェクトを取得し、そのオブジェクトのgetId()メソッドでエントリIDを取得します。

［ApplicationStoreクラス］

static ApplicationStore selectEntry()

［解説］ 端末内のiアプリを選択
［戻り値］ AppicationStoreオブジェクト（キャンセルの時はnull）

［ApplicationStoreクラス］

int getId()

［解説］ エントリIDの取得
［戻り値］ エントリID

LaunchCanvas ...❹：iアプリのバージョンアップ

Star

iアプリからiアプリのバージョンアップを行う時にはStarApplication.getThisStarApplication().getStarApplicationManager().upgrade()メソッドを使います。

Star ［StarApplicationManagerクラス］

void upgrade()

［解説］ iアプリのバージョンアップ

DoJa

iアプリからiアプリのバージョンアップを行う時も同様に、IApplicationクラスのlaunch()メソッドを使います。第1引数にIApplication.LAUNCH_VERSIONUP、第2引数にnullを指定します。

▶ADFの設定

ADFは「UseBrowser」と「LaunchApp」を次のように設定してください。

UseBrowse	launch（チェックを入れる）
LaunchApp	yes（チェックを入れる）

iアプリからブラウザを起動するには「UseBrowser」の「launch」にチェックを入れる必要があります。iアプリからiアプリを起動するには「LaunchApp」の「yes」にチェックを入れる必要があります。この設定を行わないと、起動しようとした時に例外が発生します。

4-6 ブラウザやメーラからのiアプリ起動

ブラウザやメーラからiアプリを起動する方法について説明します。「4.1 端末情報の取得」で作った「PhoneInfoEx」を起動させます。

▶ブラウザからのiアプリ起動

・ADFの設定

ADFの「LaunchByBrowser」を次のように設定してください。

LaunchByBrowser	any

ADF設定

フィルター：	
UseNetwork	☐ yes
TargetDevice	
LaunchAt	
UseTelephone	☐ call
UseBrowser	☐ launch
UseDTV	☐ launch
LaunchByMail	any
LaunchByBrowser	any
LaunchByToruCa	
LaunchByBML	
AllowPushBy	
AppTrace	☐ on
DrawArea	x
GetSysInfo	☑ yes
TrustedAPID	
TrustedLmd	

設定　取消

ブラウザからiアプリを起動するには、ADFの「LaunchByBrowser」にiアプリ起動を許可するサイトのURLを指定する必要があります。最大255バイトで、前方一致です。「any」を指定した時は無制限に許可となります。

・iアプリを起動するHTMLの作成

ブラウザからiアプリを起動するHTMLを作成します。

[launch.html]

```
<OBJECT declare id="launch"
    data="http://npaka.net/star10/star_dl/PhoneInfoEx.jam"
    type="application/x-jam">
</OBJECT>
<A ista="#launch" href="error.html">起動</A><BR>
```

OBJECT要素は、ダウンロードするiアプリのADFの指定に使います。OBJECT要素のid属性にはA要素と対応付けるための任意の名前を指定し、data属性には起動したいアプリのADFのURLを指定します。

A要素は、iアプリをダウンロードするためのリンクの指定に使います。A要素のista属性には、OBJECTタグのid属性で指定した名前に「#」を付けて指定します。A要素で指定したリンクをクリックした時、対応付けたOBJECT要素で指定しているiアプリをダウンロードします。href属性には、非対応端末からアクセスした時に開くHTMLファイルを指定します。

Column» ブラウザからのiアプリ起動時のパラメータ渡し

ブラウザからiアプリにパラメータを渡たす時は、<OBJECT>と</OBJECT>の間にPARAM要素を追加します。PARAM要素のNAME属性には属性名を、VALUE属性には値を指定します。パラメータは最大16個まで指定でき、属性と値の合計バイト数が最大255バイトまでという制限があります。

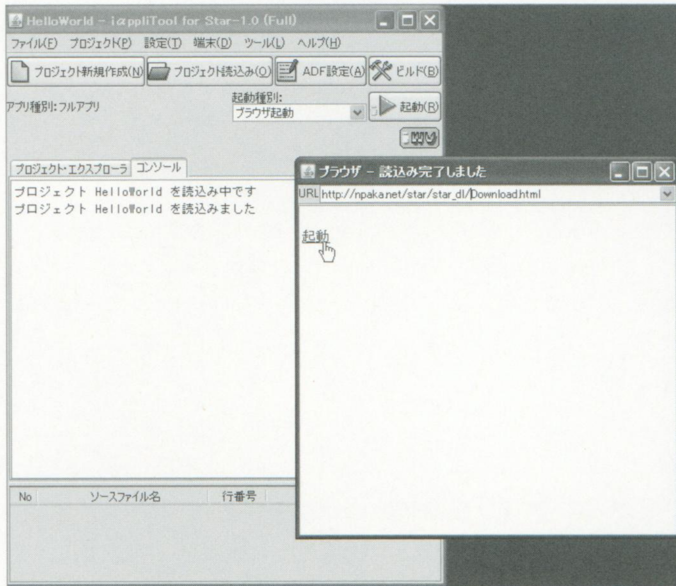
属性Param1に値「Browser」と、属性Param2に値「1」のパラメータを指定するには、次のように記述します。

```
<OBJECT declare id="launch"
        data="http://npaka.net/star10/star_dl/PhoneInfoEx.jam"
        type="application/x-jam">
<PARAM name="Param1" value="Browser">
<PARAM name="Param2" value="1">
</OBJECT>
<A ista="#launch" href="error.html">起動</A><BR>
```

iアプリで渡されたパラメータを取得するには、StarはStarApplicationManagerクラス、DoJaはIApplicationクラスのgetParameter()メソッドを使います。

・エミュレータでの実行

iαppliToolの「起動種別：」のプルダウンメニューから［ブラウザ起動］を選択して［起動］ボタンをクリックすると「ブラウザウィンドウ」が開きます。先ほど作成したHTMLをサーバにアップロードし、「URL」の項目に入力するとWebページが表示されます。そしてリンクをクリックするとiアプリが起動します。



▶メーラからのiアプリ起動

・ADFの設定

ADFの「LaunchByMail」を次のように設定してください。

LaunchByMail	any

メーラからiアプリを起動するには、ADFの「LaunchByMail」にiアプリ起動を許可するメールアドレスを指定する必要があります。最大50バイトで、後方一致です。「any」を指定した時は無制限に許可となります。

・iアプリを起動するメールの作成

iアプリを起動するメールを作成します。その内容の例は以下のようにします。

```
PhoneInfoEx を起動します。

--B:A
TEXT=" 起動 "
ADF="http://npaka.net/star10/star_dl/PhoneInfoEx.jam"
```

「--B:A」はメール本文とiアプリ起動情報の境界を示す識別子です。これ以前がメール本文、これ以降がメールからアプリを起動するための命令となります。

「TEXTキー」はリンクとして表示する文字列を指定します。
「ADFキー」は起動するiアプリのADFのURLを指定します。

・エミュレータでの実行

エミュレータで開きたいメールは、iαppli Development Kitの「`lib¥mail`」フォルダにおきます。拡張子は「`.mail`」で、初期状態では`0.mail`と`1.mail`が入っています。

`0.mail`を次のように編集し「`PhoneInfoEx.mail`」と名前を変えて保存してください。

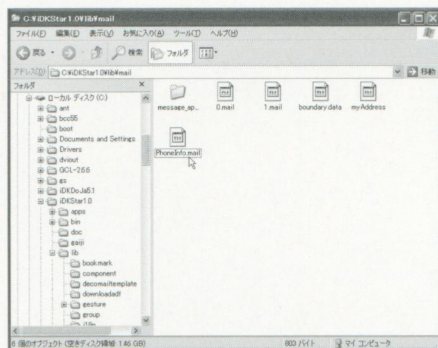
［PhoneInfoEx.mail］

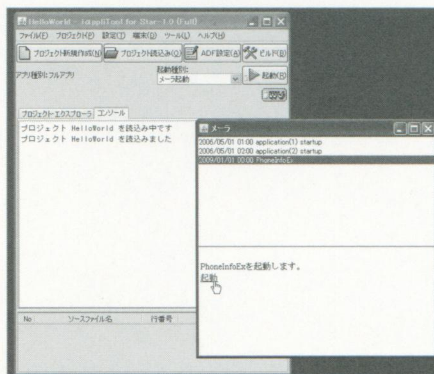
```
From: npaka@npaka.net
To: npaka@npaka.net
Subject: PhoneInfoEx
Date: 2009/01/01 00:00

PhoneInfoEx を起動します。

--B:A
TEXT=" 起動 "
ADF="http://npaka.net/star10/star_dl/PhoneInfoEx.jam"
```

iαppliToolの「起動種別：」のプルダウンメニューより［メーラ起動］を選択して［起動］ボタンをクリックします。「メーラウィンドウ」が開くので、先ほど作ったメールを選択し、リンクをクリックするとiアプリが起動します。





Column»　メーラからのiアプリ起動時のパラメータ渡し

メーラからiアプリにパラメータを渡したい時は、ADFキーの後ろに「"属性名"="値"」の書式でパラメータを追加します。パラメータは最大16個まで指定でき、属性と値の合計バイト数が最大255バイトまでという制限があります。

属性Param1に値「Mailer」と、属性Param2に値「2」のパラメータを指定するには、次のように記述します。

```
PhoneInfoEx を起動します。


--B:A
TEXT=" 起動 "
ADF="http://npaka.net/star10/star_dl/PhoneInfoEx.jam"
"Param1"="Mailer"
"Param2"="2"
```

iアプリで渡されたパラメータを取得するには、StarはStarApplicationManagerクラス、DoJaはIApplicationクラスのgetParameter()メソッドを使います。

Column»　位置情報の取得

GPS対応の携帯端末は位置情報を取得することができます。しかし、iアプリから直接位置情報を取得する機能はiアプリDXの機能となってるため、勝手アプリでは利用できません。iモードサイト経由で位置情報を取得する必要があります。

GPS対応のiモード端末では、A要素もしくはFORM要素に"lcs"という属性を追加することによって、位置情報が位置情報URLに変換され、送信されます。

```
<A HREF="http://npaka.net/star10/gps/launch.php" lcs>
```

lat（緯度：latitude）

緯度とは、赤道を基点（0度）として、北に+90°（北緯）、南に-90度（南緯）の角度で示したものです。

書式は「±dd.mm.ss.sss」。60進法で、度をdd、分をmm、秒をss として表記し、小数点以下は10進法で3桁表記します。度分秒の各パラメータは「.」で区切り、全ての記述は1バイト文字（半角文字）で記述します。

```
【例】lat = +35.00.35.6・00
```

lon（経度：longitude）

経度とは、イギリスの旧グリニッジ天文台を通る子午線を基点（0度）として、東に180度（東経）、西に180度（西経）の角度で示したものです。

書式は「±ddd.mm.ss.sss」。180進法で度をddd、60進法で分をmm、秒をss として表記し、小数点以下は10進法で3桁表記します。度分秒の各パラメータは「.」で区切り、全ての記述は1バイト文字（半角文字）で記述します。

```
【例】lon = +135.41.35.600
```

geo（測地系：datum）

測位結果の測地基準系を任意の文字列で表記したもので、GPS対応iモードでは全て（世界測地系）です。機種により「wgs84」は「WGS84」になることもあります。

```
【例】geo = wgs84
```

x-acc（測位レベル: accuracy）

測位結果の誤差範囲を測位レベルのパラメータとして表記したものです。

測位レベル	誤差範囲
3	水平誤差 < 50m
2	50m <= 水平誤差 < 300m
1	300m <= 水平誤差

詳しくは、「作ろうiモードコンテンツ-GPS」を参照してください。

・作ろうiモードコンテンツ - GPS
http://www.nttdocomo.co.jp/service/imode/make/content/gps/

スクラッチパッドと通信

5

本章では、スクラッチパッド（iアプリの端末内のデータ保存領域）やSDカードへのデータの読み書き、HTTP通信やFeliCaのアドホック通信を利用する方法について解説します。

これらの機能を活用することで、端末にハイスコアを保存したり、ネットを介した多人数のオンラインゲームの作成に活用できます。

5-1 スクラッチパッドの読み書き

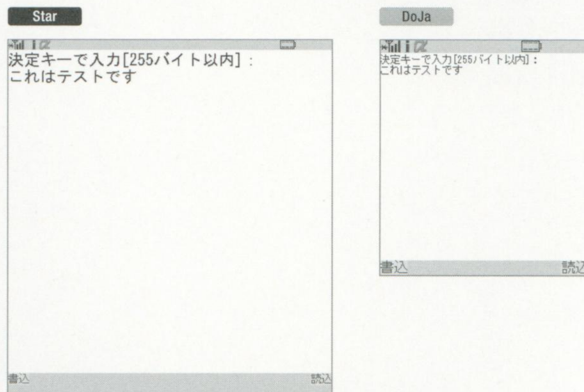


Star
http://book.mycom.co.jp/support/pc/star/d/51s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/51d.html

スクラッチパッドの読み書きを行うプログラムを作ります。



決定キー	テキスト編集
ソフトキー1	スクラッチパッド書き込み
ソフトキー2	スクラッチパッド読み込み

スクラッチパッドとは、iアプリごとに割り当てられるデータ保存領域のことです。ゲームのハイスコアなどのデータはこのスクラッチパッドに保存することで、iアプリ終了後も保持し続けることができます。実機のメニューの**バージョンアップ**によりiアプリを更新しても、スクラッチパッドの指定サイズ（SPsize）が同じであれば、中身は保持したままです。他のiアプリが持つスクラッチパッドにアクセスすることはできません。

スクラッチパッドのサイズは端末ごとに異なります。付録a-1や以下のサイトを参照してください。

- **iアプリ　端末スペック一覧**
  http://www.nttdocomo.co.jp/service/imode/make/content/spec/iappli/index.html

このプログラムは、次の2つのクラスで構成されています。

- **ScratchExクラス（ScratchEx.java）**
- **ScratchCanvasクラス（ScratchCanvas.java）**

chapter

1
2
3
4
5
6
7
8
a

プロジェクト名「**ScratchEx**」でプロジェクトを作成してください。

▶ScratchExクラス

ScratchExクラスは、プログラムの本体となるクラスです。

Star ScratchEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// スクラッチパッドの読み書き（本体）
public class ScratchEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {        // ⓒ
        ScratchCanvas c = new ScratchCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa ScratchEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class ScratchEx extends IApplication {
```

ⓒ

```
  public void start() {
```

▶ScratchCanvasクラス

ScratchCanvasクラスは、キャンバスとなるクラスです。

Star ScratchCanvas.java

```
import com.docomostar.ui.Canvas;    // ⓐ
import com.docomostar.ui.Dialog;    //
import com.docomostar.ui.Display;   //
```

```
import com.docomostar.ui.Graphics;  //
import com.docomostar.ui.TextBox;   //
import javax.microedition.io.Connector;
import java.io.InputStream;
import java.io.OutputStream;

// スクラッチパッドの読み書き ( キャンバス )
public class ScratchCanvas extends Canvas
    implements Runnable {
    private int    keyEvent = -999;   // キーイベント
    private String text     = "";     // テキスト

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        // ソフトラベル
        setSoftLabel(SOFT_KEY_1," 書込 ");
        setSoftLabel(SOFT_KEY_2," 読込 ");

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString(" 決定キーで入力 [255 バイト以内 ] : ",0,23*1); // ⓑ
            g.drawString(text,0,24*2);                                //
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_SELECT) {
                // 文字入力
                imeOn(text,TextBox.DISPLAY_ANY,TextBox.KANA);
            } else if (keyEvent==Display.KEY_SOFT1) {
                // スクラッチパッドへの書き込み
                writeScratch();
            } else if (keyEvent==Display.KEY_SOFT2) {
                // スクラッチパッドからの読み込み
                readScratch();
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
```

Next page. ↓

chapter
1
2
3
4
5
6
7
8
a

```
        }
    }

    // スクラッチパッドへの書き込み ...❶
    void writeScratch() {  // ⓒ
        int    size;
        byte[] data;
        OutputStream out = null;
        try {
            // スクラッチパッドと接続
            out = Connector.openOutputStream("scratchpad:///0;pos=0");
                                                        // ...❶-1
            // スクラッチパッドへの書き込み
            data = text.getBytes();
            size = data.length;
            out.write(size);
            if (size>0) out.write(data);

            // スクラッチパッドと切断
            out.close();
        } catch (Exception e) {
            // 例外処理
            try {
                if (out != null) out.close();
            } catch (Exception e2) {
            }
            showDialog("書き込み失敗");
        }
    }

    // スクラッチパッドからの読み込み ...❷
    void readScratch() {
        int    size;
        byte[] data;
        InputStream in = null;
        try {
            // スクラッチパッドと接続
            in = Connector.openInputStream("scratchpad:///0;pos=0");
                                                        // ...❷-2
            // スクラッチパッドからの読み込み
            size = (int)(in.read()&0xFF);
            data = new byte[size];
            if (size>0) {
                in.read(data);
                text = new String(data);
            }

            // スクラッチパッドと切断
            in.close();
```

```
        } catch (Exception e) {
            // 例外処理
            try {
                if (in != null) in.close();
            } catch (Exception e2) {
            }
            showDialog(" 読み込み失敗 ");
        }
    }

    // ダイアログの表示
    void showDialog(String text) {                                  // ⓓ
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR," エラー "); //
        dialog.setText(text);                                       //
        dialog.show();
    }

    // IME イベントの処理
    public void processIMEEvent(int type,String text) {
        if (type==IME_COMMITTED) {
            if (text.getBytes().length<=255) {
                this.text = text;
            } else {
                showDialog("255 バイト以内にして下さい ");
            }
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa ScratchCanvas.java

ⓐ

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.TextBox;
```

ⓑ

```
            g.drawString("決定キーで入力[255バイト以内]：",0,12);
            g.drawString(text,0,24);
```

ⓒ

```
    private void writeScratch() {
```

ⓓ

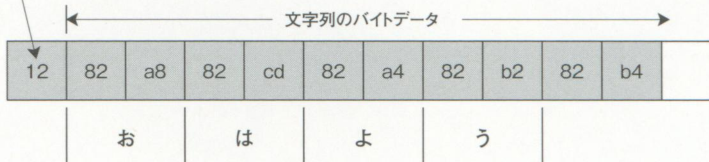
```
    private void showDialog(String text) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー");
        dialog.setFont(Font.getFont(Font.SIZE_SMALL));
```

書き込むバイトデータのフォーマット

スクラッチパッドにはデータを**バイトデータ**として保存します。スクラッチパッドの先頭より何バイト目から読み込み、何バイト目から書き込むというシンプルな操作しかできません。文字列の読み書きだけでも、その文字列が何バイトなのかという情報がなければ、どこまで読み込んでよいのかもわかりません。そのため、どんなフォーマットで書き込むかを決めておく必要があります。

今回は書き込むバイトデータのフォーマットを次のように決めました。

0バイト目	文字列のバイトサイズ（0～255）
1～255バイト目	文字列のバイトデータ



ScratchCanvas ...❶：スクラッチパッドへの書き込み

スクラッチパッドと接続するには、javax.microedition.ioパッケージの**Connectorクラス**を使います。openOutputStream()メソッドを使って、java.ioパッケージのOutputStreamオブジェクトを取得してください。

［Connectorクラス］

static OutputStream openOutputStream(*location*)

［解説］　出力ストリームの接続
［引数］　*location*　書き込み先：String型
［戻り値］　出力ストリーム

引数の*location*（書き込み先）には、次の書式で指定します。

```
scratchpad:///0;pos=先頭からのバイト数
```

スクラッチパッドの先頭から0バイト目から書き込むには、次のように記述します。

［ScratchCanvas.java］...❶-1

```
out = Connector.openOutputStream("scratchpad:///0;pos=0");
```

0バイト目からの時は、「;pos=0」を省略することもできます。

```
out = Connector.openOutputStream("scratchpad:///0");
```

スクラッチパッドへデータを書き込むには、OutputStreamクラスのwrite()メソッドを使います。

［OutputStreamクラス］

void write(*data, off, len*)

［解説］　バイトデータの書き込み
［引数］　*data*　バイトデータ：byte[]型
　　　　　off　データの開始オフセット：int型
　　　　　len　書き込むバイト数：int型

引数*data*（バイトデータ）の*off*番目から（*off* + *len* - 1）番目のデータを書き込みます。

［OutputStreamクラス］

void write(*data*)

［解説］ バイトデータの書き込み
［引数］ *data* バイトデータ：byte[]型

引数*data*（バイトデータ）の全データ（0番目から（*data.length* - 1)番目のデータ）を書き込みます。

［OutputStreamクラス］

void write(*data*)

［解説］ バイトデータの書き込み
［引数］ *data* バイトデータ（0～255）：int型

1バイトのバイトデータを書き込みます。引数はint型ですが、0～255の値を指定します。

StringクラスのgetBytes()メソッドでバイトデータを取得し、それのlengthでバイトサイズを取得しています。

［Stringクラス］

byte[] getBytes()

［解説］ バイトデータの取得
［戻り値］ バイトデータ

スクラッチパッドと切断するにはclose()メソッドを使います。

［OutputStreamクラス］

void close()

［解説］ 出力ストリームの切断

切断しないと、iアプリを終了させるまでスクラッチパッドに接続できなくなるので、例外が発生した時も切断するようにしてください。

ScratchCanvas ...❷：スクラッチパッドからの読み込み

スクラッチパッドと接続するには、**Connectorクラス**を使います。openInputStream()メソッドを使って、InputStreamオブジェクトを取得してください。

［Connectorクラス］

static InputStream openInputStream(*location*)

［解説］　入力ストリームの接続
［引数］　*location*　読み込み先：String型
［戻り値］　入力ストリーム

引数*location*（読み込み先）の書式は、openOutputStream()メソッドの引数*location*（書き込み先）の書式と同じです。

スクラッチパッドの先頭から0バイト目から読み込むには、次のように記述します。

ScratchCanvas.java ...❷-1

```
in = Connector.openInputStream("scratchpad:///0;pos=0");
```

スクラッチパッドからデータを読み込むには、**InputStreamクラス**のread()メソッドを使います。

［InputStreamクラス］

void read(*data, off, len*)

［解説］　バイトデータの読み込み
［引数］　*data*　バイトデータ：byte[]型
　　　　off　データの開始オフセット：int型
　　　　len　読み込むバイト数：int型

引数*data*（バイトデータ）の*off*番目から（*off* + *len* - 1）番目のデータに読み込んだデータをセットします。引数*off*と*len*は、スクラッチパッドから読み込む場所を指定しているわけではないので注意してください。

[InputStreamクラス]

byte[] read(*data*)

［解説］ バイトデータの読み込み
［引数］ *data*　　バイトデータ:byte[]型

引数*data*（バイトデータ）の0番目から（*data.length* - 1）番目のデータに読み込んだデータをセットします。

[InputStreamクラス]

int read()

［解説］　バイトデータの読み込み
［戻り値］ バイトデータ（データがない時は-1）

1バイト分データを読み込みます。読み込むデータがない時は「-1」を返します。
スクラッチパッドと切断するにはclose()メソッドを使います。

[InputStreamクラス]

void close()

［解説］ 入力ストリームの切断

▶ADFの設定

ADFの「SPSize」を次のように設定してください。

SPsize	256

スクラッチパッドを使う時は「SPsize」に何バイトの保存領域を使うかを指定します。

エミュレータでスクラッチパッドの中身を見たい時は、iαppliToolのメニュー［ツール］－［スクラッチパッド...］を選んでください。16進数で表示されます。中身を削除したい時は、メニュー［ツール］－［スクラッチパッドを全て初期化］を指定します。実機ではアプリ削除時にスクラッチパッドの中身も削除されます。



Column>>スクラッチパッドの分割管理

iアプリに割り当てられたスクラッチパッドのサイズはSPsizeの指定により最大16個まで分割管理することができます。SPsizeに個々のバイトをカンマ区切りで指定します。1024バイトと2048バイトの2つに分割して使うには、次のように指定します

SPsize	1024, 2048

1つ目の領域にアクセスするには「scratchpad:///0」、2つ目の領域にアクセスするには「scratchpad:///1」を指定します。

5-2 SDカードの読み書き



Star
http://book.mycom.co.jp/support/pc/star/52s.html

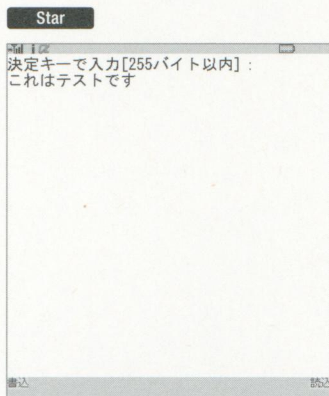


DoJa
http://book.mycom.co.jp/support/pc/star/d/52d.html

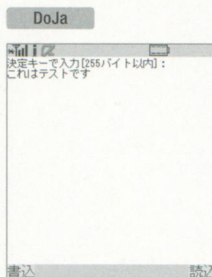
SDカードの読み書きを行うプログラムを作ります。

決定キー	テキスト編集
ソフトキー 1	SDカード書き込み
ソフトキー 2	SDカード読み込み

Star



DoJa



SD（Secure Digital）カードとはメモリーカードの規格の1つで、携帯電話やカメラなどの補助記憶装置として利用されています。著作権保護機能を搭載しているのが特徴です。SDカードはおおまかに次の3種類に分けられます。

- SDカード（32 x 24 x 2 mm）
- miniSD（21.5 x 20x 1.4 mm）
- microSD（11 x 15 x 1 mm）

違いは大きさだけで中身は同じです。アダプタでminiSDとmicroSDをSDカードとして利用することもできます。本書ではこれらをまとめてSDカードと呼ぶことにします。

iアプリからは10M程度のデータを保存できますが、アクセス時間がかかるので注意が必要です。また、エミュレータでは開発ツールのインストールフォルダの「`lib¥storagedevice¥ext0`」フォルダ下に保存されます。

このプログラムは、次の2つのクラスで構成されています。

- **StorageDeviceEx**クラス（**StorageDeviceEx.java**）
- **StorageDeviceCanvas**クラス（**StorageDeviceCanvas.java**）

プロジェクト名「**StorageDeviceEx**」でプロジェクトを作成してください。

▶StorageDeviceExクラス

StorageDeviceExクラスは、プログラムの本体となるクラスです。

Star StorageDeviceEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// SD カードの読み書き ( 本体 )
public class StorageDeviceEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {              // ⓒ
        StorageDeviceCanvas c = new StorageDeviceCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa StorageDeviceEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class StorageDeviceEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶StorageDeviceCanvasクラス

StorageDeviceCanvasクラスは、キャンバスとなるクラスです。

Star StorageDeviceCanvas.java

```
import com.docomostar.device.StorageDevice;  // ⓐ
import com.docomostar.fs.AccessToken;        //
import com.docomostar.fs.File;               //
import com.docomostar.fs.StarStorageService; //
import com.docomostar.fs.StarAccessToken;    //
import com.docomostar.io.FileEntity;         //
import com.docomostar.ui.Canvas;             //
import com.docomostar.ui.Dialog;             //
import com.docomostar.ui.Display;            //
import com.docomostar.ui.Graphics;           //
import com.docomostar.ui.TextBox;            //
import javax.microedition.io.Connector;      //
import java.io.InputStream;
import java.io.OutputStream;

// SDカードの読み書き(キャンバス)
public class StorageDeviceCanvas extends Canvas
    implements Runnable {
    private int    keyEvent = -999; // キーイベント
    private String text     = "";   // テキスト

    // 処理
    public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        // ソフトラベル
        setSoftLabel(SOFT_KEY_1,"書込");
        setSoftLabel(SOFT_KEY_2,"読込");

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("決定キーで入力[255バイト以内]:",0,24*1); // ⓑ
            g.drawString(text,0,24*2);                           //
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_SELECT) {
```

```
                // 文字入力
                imeOn(text,TextBox.DISPLAY_ANY,TextBox.KANA);
            } else if (keyEvent==Display.KEY_SOFT1) {
                // SD カードへの書き込み
                writeStorage();
            } else if (keyEvent==Display.KEY_SOFT2) {
                // SD カードからの読み込み
                readStorage();
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // SD カードへの書き込み
    void writeStorage() { // ⓒ
        int    size;
        byte[] data;
        File         file   = null;
        FileEntity   entity = null;
        OutputStream out    = null;

        // アクセス権の指定 ...❶
        AccessToken dat = StarStorageService.getAccessToken(          // ⓓ
            StarAccessToken.ACCESS_UIM,                 // アクセス識別子
            StarStorageService.SHARE_APPLICATION);  // 共有識別子

        try {
            // SD カードとの接続 ...❷
            StorageDevice sd = StorageDevice.getInstance("/ext0");

            // ファイルの取得と生成 ...❸
            try {
                file = sd.getFolder(dat).getFile("test.txt");
            } catch (Exception e2) {
                file = sd.getFolder(dat).createFile("test.txt");
            }

            // ファイルへの書き込み ...❹
            data = text.getBytes();
            size = data.length;
            entity = file.open(File.MODE_WRITE_ONLY);
            out = entity.openOutputStream();
            if (size>0) out.write(text.getBytes());
```

Next page.

chapter

1
2
3
4
5
6
7
8
a

```
            out.close();
            entity.close();
        } catch (Exception e) {
            // 例外処理
            try {
                if (out    != null) out.close();
                if (entity != null) entity.close();
            } catch (Exception e2) {
            }
            showDialog("書き込み失敗");
        }
    }

    // SDカードからの読み込み
    void readStorage() { // ⓔ
        int    size;
        byte[] data;
        FileEntity  entity = null;
        InputStream in     =null;

        // アクセス権の指定
        AccessToken dat = StarStorageService.getAccessToken(          // ⓕ
            StarAccessToken.ACCESS_UIM,             // アクセス識別子
            StarStorageService.SHARE_APPLICATION); // 共有識別子

        try {
            // SDカードとの接続
            StorageDevice sd = StorageDevice.getInstance("/ext0");

            // ファイルの取得
            File file = sd.getFolder(dat).getFile("test.txt");

            // ファイルからの読み込み …❺
            entity = file.open(File.MODE_READ_ONLY);
            in = entity.openInputStream();
            data = new byte[1024];
            size = in.read(data);
            text = new String(data,0,size);
            in.close();
            entity.close();
        } catch (Exception e) {
            // 例外処理
            try {
                if (in     != null) in.close();
                if (entity != null) entity.close();
            } catch (Exception e2) {
            }
            showDialog("読み込み失敗");
        }
```

```
    }

    // ダイアログの表示
    void showDialog(String text) {                                    // ⑨
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー"); //

        dialog.setText(text);
        dialog.show();
    }

    // IMEイベントの処理
    public void processIMEEvent(int type,String text) {
        if (type==IME_COMMITTED) {
            if (text.getBytes().length<=255) {
                this.text = text;
            } else {
                showDialog("255バイト以内にして下さい");
            }
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa StorageDeviceCanvas.java

ⓐ

```
import com.nttdocomo.device.StorageDevice;
import com.nttdocomo.fs.dojaAccessToken;
import com.nttdocomo.fs.dojaStorageService;
import com.nttdocomo.fs.File;
import com.nttdocomo.io.FileEntity;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.TextBox;
```

chapter

1

2

3

4

5

6

7

8

a

ⓑ

```
            g.drawString("決定キーで入力[255バイト以内]：",0,12);
            g.drawString(text,0,24);
```

ⓒ

```
    private void writeStorage() {
```

ⓓ

```
        // アクセス権
        dojaAccessToken dat = dojaStorageService.getAccessToken(
            dojaAccessToken.ACCESS_UIM,               // アクセス識別子
            dojaStorageService.SHARE_APPLICATION);  // 共有識別子
```

ⓔ

```
    // SDカードからの読み込み
    private void readStorage() {
```

ⓕ

```
        // アクセス権
        dojaAccessToken dat = dojaStorageService.getAccessToken(
            dojaAccessToken.ACCESS_UIM,              // アクセス識別子
            dojaStorageService.SHARE_APPLICATION); // 共有識別子
```

ⓖ

```
    // ダイアログの表示
    private void showDialog(String text) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー");
        dialog.setFont(Font.getFont(Font.SIZE_SMALL));
```

StorageDeviceCanvas ...❶：アクセス権の指定

SDカードにアクセスするには**アクセス権**を指定する必要があります。アクセス権には**アクセス識別子**と**共有識別子**があります。

アクセス識別子とは、SDカードに保存されたデータが、保存された時と異なる機器構成でも利用できるかどうかを指定するものです。具体的には次の3つの定数の論理和を指定します。

StarAccessToken.ACCESS_PLATFORM	保存時と同じ端末の時のみアクセス可能
StarAccessToken.ACCESS_SERIES	保存時と同じシリーズ機種の時のみアクセス可能
StarAccessToken.ACCESS_UIM	保存時と同じUIMカードの時のみアクセス可能

共有識別子とは、保存したiアプリ以外のiアプリでも利用できるかどうかを指定するものです。具体的には次の2つの定数を指定します。

StarStorageService.SHARE_APPLICATION	保存時と同じiアプリのみアクセス可能
StarStorageService.SHARE_CONTENTS_PROVIDER	保存時と同じコンテンツプロバイダのみアクセス可能

StarStorageServiceクラスのgetAccessToken()メソッドを使って、2つの識別子を保持するStarAccessTokenオブジェクトを生成します。

［StarStorageServiceクラス］

static StarAccessToken getAccessToken(*access, share***)**

［解説］ DoJaAccessTokenオブジェクトの生成
［引数］ *access* アクセス識別子：int型
share 共有識別子：int型
［戻り値］ アクセス権

DoJa DoJaStorageServiceクラスのgetAccessToken()メソッドを使ってDoJaAccessTokenオブジェクトを生成します。

StorageDeviceCanvas ...❷：SDカードとの接続

SDカードとの接続を行うには、**StorageDeviceクラス**のgetInstance()メソッドを使います。

［StorageDeviceクラス］

static StorageDevice getInstance(*deviceName***)**

［解説］ SDカードと接続
［引数］ *deviceName* デバイス名：String型
［戻り値］ ストレージデバイス

引数の*deviceName*（デバイス名）には「/ext0」を指定します。

StorageDeviceCanvas …❸：ファイルの取得と生成

SDカード内のファイルを取得するには、まずgetFolder()メソッドでフォルダを取得し、その後getFile()メソッドでファイルを取得します。

［StorageDeviceクラス］

Folder getFolder(*accessToken*)

［解説］　フォルダの取得
［引数］　*accessToken*　AccessTokenオブジェクト：AccessToken型
［戻り値］　フォルダ

［Folderクラス］

File getFile(*fileName*)

［解説］　ファイルの取得
［引数］　*fileName*　ファイル名：String型
［戻り値］　ファイル

ファイルの取得に失敗した時は、次のような例外が投げられます。

MediaNotFoundException	SDカードが存在しない時
FileNotAccessibleException	ファイルが存在しない時、取得できない時

ファイルが存在しない時は生成します。ファイルを生成するにはcreateFile()メソッドを使います。

［Folderクラス］

File createFile(*fileName*)

［解説］　ファイルの生成
［引数］　*fileName*　ファイル名：String型
［戻り値］　ファイル

ファイルの生成に失敗した時は、次のような例外が投げられます。

MediaNotFoundException	SDカードが存在しない時
FileNotAccessibleException	ファイルの生成に失敗した時
FileSystemFullException	空き容量がない時

StorageDeviceCanvas ...❹：ファイルへの書き込み

ファイルと接続するには、**File**クラスのopen()メソッドでFileEntityオブジェクトを取得し、**FileEntity**クラスのopenOutputStream()メソッドでOutputStreamオブジェクトを取得してください。

［Fileクラス］

FileEntity open(*mode*)

［解説］　ファイルとの接続
［引数］　*mode*　モード：int型
［戻り値］　FileEntityオブジェクト

引数*mode*には次の定数を指定します。

File.MODE_READ_ONLY	読み込みのみ
File.MODE_READ_WRITE	読み書き
File.MODE_WRITE_ONLY	書き込みのみ

［FileEntityクラス］

OutputStream openOutputStream()

［解説］　出力ストリームの取得
［戻り値］　出力ストリーム

ファイルへのデータの書き込みは、スクラッチパッドと同様に**OutputStream**を使います。ファイルと切断するにはOutputStreamオブジェクトとFileEntityオブジェクトのclose()メソッドを呼んでください。切断しないと、iアプリを終了させるまでSDカードにアクセスできなくなるので、例外が発生した時も切断するようにします。

StorageDeviceCanvas …❺：ファイルからの読み込み

ファイルと接続するには、Fileクラスのopen()メソッドでFileEntityオブジェクトを取得し、**FileEntityクラス**のopenInputStream()メソッドでInputStreamオブジェクトを取得してください。

［FileEntityクラス］

InputStream openInputStream()

［解説］　入力ストリームの取得
［戻り値］　入力ストリーム

ファイルからのデータの読み込みは、スクラッチパッドと同様にInputStreamを使います。ファイルと切断するにはInputStreamオブジェクトとFileEntityオブジェクトのclose()メソッドを呼んでください。

▶ADFの設定

ADFの「UseStorage」を次のように設定してください。

UseStorage	ext

SDカードを使う時は「UseStorage」の「ext」にチェックを入れる必要があります。

Column»　保存データの暗号化

SDカードに保存したデータを解読されないように暗号化するには、ファイル属性指定付きのcreateFile()メソッドを使います。

［Folderクラス］

File createFile(*fileName, attributes*)

［解説］　ファイルの生成
［引数］　*fileName*　ファイル名：String型
　　　　attributes　ファイル属性：FileAttribute[]型
［戻り値］　ファイル

FileAttributeクラスはファイル属性情報を保持するクラスの親クラスで、SDカードの場合は、FileAttributeクラスを継承したEncryptionAttributeをさらに継承し

たSDBindingEncryptionAttributeクラス（com.nttdocomo.fs.sdパッケージ）を利用します。暗号化では、次のsetBlockSize()メソッドとsetEncryption()メソッドを利用します。

［SDBindingEncryptionAttributeクラス］

void setBlockSize(*blockSize*)

［解説］暗号化ブロックサイズの指定
［引数］*blockSize* 暗号化ブロックサイズ：int型

引数の*blockSize*（暗号化ブロックサイズ）には、以下の条件を満たした値を指定してください。

```
64 << E
（E = 0, 1, …, 13）
```

［EncryptionAttributeクラス］

void setEncryption(*encryption*)

［解説］暗号化のON/OFFの指定
［引数］*encryption* 暗号化のON/OFF：boolean型

［例］

```
FileAttribute[] attr = new FileAttribute[1];
attr[0] = new SDBindingEncryptionAttribute();
((SDBindingEncryptionAttribute)attr[0]).setBlockSize(64<<6);
((SDBindingEncryptionAttribute)attr[0]).setEncryption(true);
file = folder.createFile(fileName,attr);
```

暗号化機能を有効にするとアクセス速度は低下します。また、書き込む内容より暗号化ブロックサイズが大きい場合はアクセス速度が低下します。

chapter
1
2
3
4
5
6
7
8
a

5-3 ネットからのデータ読み込み

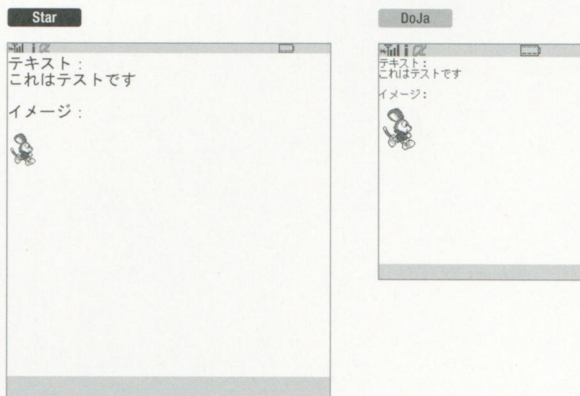


Star
http://book.mycom.co.jp/support/pc/star/53s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/53d.html

ネットからテキストやイメージを読み込むiアプリを作ります。ユーザが端末設定で通信許可していない時は、通信できません。



iアプリで使える通信プロトコルは**HTTP**と**HTTPS**（HTTPにデータ暗号化機能を付けたもの）です。さらにセキュリティのため、iアプリから通信を行う相手は、そのiアプリのダウンロード場所と同じURL、ホスト、ポート番号でなければなりません。URLを数値アドレスで指定すること（`127.0.0.1`など）もできません。

HTTPは、「`http://`」で始まるURLのリソースにアクセスするためのプロトコルです。HTTPで通信するには、**GET**と**POST**の2種類の方法があります。単純にテキストファイルやイメージファイルをダウンロードするにはGETを使い、クライアント側からサーバに情報をアップロードするにはPOSTを使います。

iアプリDXであればダウンロード元以外のサーバとの通信や、TCP/UDPによるソケット通信も行えます。

このプログラムは、次の2つのクラスで構成されています。

- **HttpExクラス（HttpEx.java）**
- **HttpCanvasクラス（HttpCanvas.java）**

プロジェクト名「**HttpEx**」でプロジェクトを作成してください。

▶テキストファイルとイメージファイルの準備

ネット上から読み込むテキストファイルとイメージファイルを用意します。ネット上のiアプリを公開する同じ場所に配置してください。

・テキストファイル：`test.txt`（文字コードはshift-JIS）

```
これはテストです
```

・イメージファイル：`jyagi.gif`（48x48ドット）



▶HttpExクラス

HttpExクラスは、プログラムの本体となるクラスです。

Star HttpEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// ネットからテキストやイメージを読み込む ( 本体 )
public class HttpEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {     // ⓒ
        HttpCanvas c = new HttpCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa HttpEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class HttpEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶HttpCanvasクラス

HttpCanvasクラスは、キャンバスとなるクラスです。

Star HttpCanvas.java

```
import com.docomostar.StarApplicationManager; // ⓐ
import com.docomostar.io.HttpConnection;      //
import com.docomostar.media.MediaImage;       //
import com.docomostar.media.MediaManager;     //
import com.docomostar.ui.Canvas;              //
import com.docomostar.ui.Dialog;              //
import com.docomostar.ui.Graphics;            //
import com.docomostar.ui.Image;               //
import javax.microedition.io.Connector;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;

// ネットからのデータ読み込み(キャンバス)
public class HttpCanvas extends Canvas
    implements Runnable {

    // 処理
    public void run() {
        // グラフィック
        Graphics g = getGraphics();

        try {
            // ネットからのデータ読み込み
            String text = readText(
                StarApplicationManager.getSourceURL()+  // ⓑ
                "test.txt");
            Image image = readImage(
                StarApplicationManager.getSourceURL()+  // ⓒ
                "jyagi.gif");

            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
```

```
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("テキスト：",0,24*1);  // ⓓ
            g.drawString(text,0,24*2);          //
            g.drawString("イメージ：",0,24*4);  //
            g.drawImage(image,0,24*5);          // ⓓ
            g.unlock(true);
        } catch (Exception e) {
            showDialog("通信失敗");
        }
    }

    // ネットからのバイトデータの読み込み ...❶
    private byte[] readByte(String url) throws Exception {
        HttpConnection        c   = null;
        InputStream           in  = null;
        ByteArrayOutputStream out = null;
        try {
            // 接続
            c = (HttpConnection)Connector.open(url,Connector.READ,true);
            c.setRequestMethod(HttpConnection.GET);
            c.connect();
            in = c.openInputStream();
            out = new ByteArrayOutputStream();

            // 読み込み
            byte[] w = new byte[10240];
            while (true) {
                int size = in.read(w);
                if (size <= 0) break;
                out.write(w,0,size);
            }

            // 切断
            out.close();
            in.close();
            c.close();
            return out.toByteArray();
        } catch (Exception e) {
            // 例外処理
            try {
                if (out != null) c.close();
                if (in  != null) in.close();
                if (c   != null) c.close();
            } catch (Exception e2) {
            }
            throw e;
        }
```

Next page. ⬇

chapter

1

2

3

4

5

6

7

8

a

```
    }

    // テキストの読み込み
    private String readText(String url) throws Exception {
        // バイトデータのテキストへの変換 ...❷
        byte[] data = readByte(url);
        return new String(data);
    }

    // イメージの読み込み
    private Image readImage(String url) throws Exception {
        // バイトデータのイメージへの変換 ...❸
        byte[] data  = readByte(url);
        MediaImage m = MediaManager.getImage(data);
        m.use();
        return m.getImage();
    }

    // ダイアログの表示
    private void showDialog(String text) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR," エラー ");
        dialog.setText(text);
        dialog.show();
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa HttpCanvas.java

ⓐ

```
import com.nttdocomo.io.HttpConnection;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.MediaManager;
```

ⓑ

```
                IApplication.getCurrentApp().getSourceURL()+
```

ⓒ

```
                IApplication.getCurrentApp().getSourceURL()+
                "jyagi.gif");
```

ⓓ

```
            g.drawString("テキスト：",0,12*1);
            g.drawString(text,0,12*2);
            g.drawString("イメージ：",0,12*4);
            g.drawImage(image,0,12*5);
```

HttpCanvas ...❶：ネットからのバイトデータの読み込み

ネットと接続するには、com.nttdocomo.ioパッケージの**Connectorクラス**を使います。open()メソッドを使って、HttpConnectionオブジェクトを取得してください。

［Connectorクラス］

static Connection open(*name, mode, timeouts***)**

［解説］ 接続を行う

［引数］ *name* 接続先URL：String型

mode アクセスモード：int型

timeout タイムアウト例外が必要か：boolean型

GETを使う時は、引数の*mode*（アクセスモード）にConnector.READを指定します。HttpConnectionクラスのオブジェクトが取得できたら、setRequestMethod()メソッドでHttpConnection.GETを指定します。

次に、HttpConnectionオブジェクトのconnect()メソッドを呼んだ後、openInputStream()メソッドを使って、InputStreamオブジェクトを取得します。

データを読み込むには、**InputStreamクラス**のread()メソッドを使います。

ネットと切断するには、InputStreamクラスとHttpConnectionクラスのclose()メソッドを使います。切断しないと、iアプリを終了させるまでネットに接続できなくなるので、例外が発生した時も切断するようにします。

HttpCanvas ...❷：バイトデータのテキストへの変換

バイトデータをテキストに変換するには、**Stringクラス**のコンストラクタを使います。

HttpCanvas ...❸：バイトデータのイメージへの変換

バイトデータをイメージに変換するには、**MediaManagerクラス**のgetImage()メソッドを使います。

［MediaManagerクラス］

static MediaImage getImage(*data*)

［解説］　イメージオブジェクトの取得
［引数］　*data*　　バイトデータ：byte[]型
［戻り値］　MediaImageオブジェクト

085ページのgetImage()メソッドも参照してください。

今回は利用していませんが、バイトデータをサウンドに変換するには、MediaManagerクラスのgetSound()メソッドを使います。

［MediaManagerクラス］

static MediaSound getSound(*data*)

［解説］　サウンドオブジェクトの取得
［引数］　*data*　　バイトデータ：byte[]型
［戻り値］　MediaSoundオブジェクト

116ページのgetSound()メソッドも参照してください。

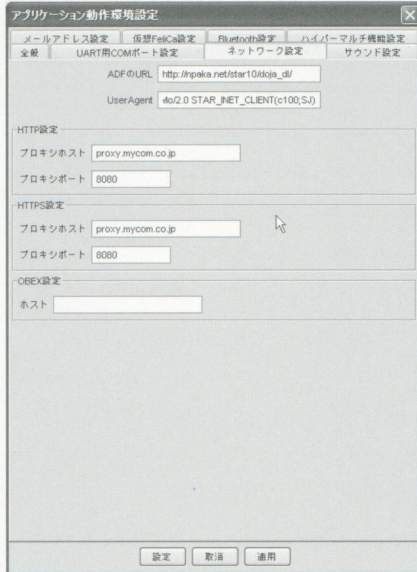
▶ADFの設定

ADFの「UseNetwork」を次のように設定してください。

UseNetwork	http

HTTP通信を行う時は「UseNetwork」の「http」にチェックを入れておく必要があります。

エミュレータで通信機能を使ったiアプリを実行する時には、ダウンロード元サーバがどこであるかを設定しておく必要があります。エミュレータ上のiアプリのダウンロードURLを設定するには、iαppliToolのメニューの［設定］−［アプリケーション動作環境設定...］で開かれるダイアログの［ネットワーク設定］タブを選択し、「ADFのURL」に指定します。今回は「http://npaka.net/star10/doja_dl/」と指定します。



Column» POST

パラメータを付加するとURL長が256バイト以上になる時や、イメージデータなどのバイトデータを送りたい時は、GETでなくPOSTを使います。**Connectorクラス**のopen()メソッドを呼ぶ時、POSTを使うには、引数の*mode*（アクセスモード）にConnector.READ_WRITEを指定します。HttpConnectionクラスのオブジェクトが取得できたら、setRequestMethod()メソッドでHttpConnection.POSTを指定します。HttpConnectionクラスのgetOutputStream()メソッドで出力ストリームを取得し、そこから送信データを送ります。

［例］

```
c = (HttpConnection)Connector.open(url, Connector.READ_WRITE,true);
c.setRequestMethod(HttpConnection.POST);
out = c.openOutputStream();
out.write(param.getBytes());
out.close();
c.connect();
in = c.openInputStream();
size = in.read(work);
in.close();
c.close();
```

5-4 JARファイルからのデータ読み込み



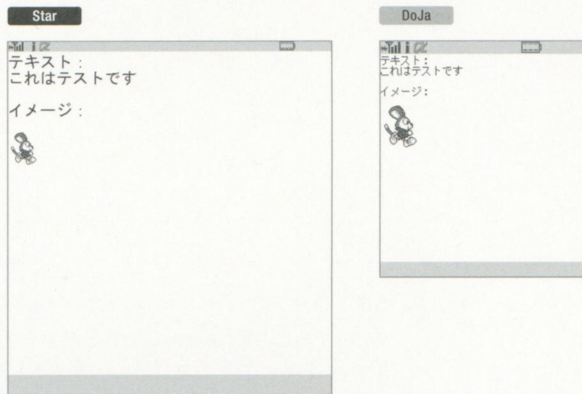
Star
http://book.mycom.co.jp/support/pc/star/54s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/54d.html

JARファイルからデータを読み込むプログラムを作ります。

JARファイルはiアプリの実行ファイル形式でもありますが、複数のファイルを１つにまとめるのが本来の役割です。これを活用することにより、複数のファイルを圧縮して一度にダウンロードすることができます。



このプログラムは、次の2つのクラスで構成されています。

- **JarInflaterEx**クラス（**JarInflaterEx.java**）
- **JarInflaterCanvas**クラス（**JarInflaterCanvas.java**）

プロジェクト名「**JarInflaterEx**」でプロジェクトを作成してください。

▶ネットから読み込むJARファイルの準備

テキストとイメージをまとめてJARファイル「test.jar」を生成し、**ネット上のiアプリを公開するのと同じ場所に配置**してください。

・テキストファイル：test.txt（文字コードはShiftJIS）

```
これはテストです
```

・イメージファイル：jyagi.gif（48x48ドット）



JARファイルを生成するには、**コマンドプロンプト**でJARコマンドを使用します。Windowsのスタートメニュー[すべてのプログラム]－[アクセサリ]－[コマンドプロンプト]を選ぶと、コマンドプロンプトが起動します。

JDKのbinフォルダにパスが通ってない時は、次のコマンドでパスを通してください。

```
set path=<JDKのbinフォルダのパス>;%path%
```

JDKのbinフォルダのパスが「C:¥Program Files¥Java¥j2re1.4.2_19¥bin¥」の時は、次のように入力します。

```
set path=C:¥Program Files¥Java¥j2re1.4.2_19¥bin¥;%path%
```

指定したフォルダ内のファイルを取り込むんでJARファイルを生成するコマンドは、次の通りです。

```
jar cMf <JARファイル名> -C <フォルダ名> .
```

「c」は新規作成、「M」はマニフェストファイルを作成しない、「f」はファイル名、「-C」はファイルを取り込むフォルダを指定するオプションです。

JARファイルはデフォルトで圧縮がかかりますが、GIFのように最初から圧縮のかかっているファイルを圧縮すると逆に増えてしまいます。圧縮しないで格納のみを行いたい時は、オプションに「0（ゼロ）」を付加してください。

```
jar cMf0 <JARファイル名> -C <フォルダ名> .
```

workフォルダにテキストとイメージを入れて、それをまとめて「test.jar」を作るコマンドは、次の通りです。

```
jar cMf test.jar -C work .
```

成功すると、コマンドプロンプトのカレントフォルダに「test.jar」が生成されます。

▶JarInflaterExクラス

JarInflaterExクラスは、プログラムの本体となるクラスです。

Star JarInflaterEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// JARファイルからテキストやイメージを読み込む(本体)
public class JarInflaterEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {           // ⓒ
        JarInflaterCanvas c = new JarInflaterCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa JarInflaterEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class JarInflaterEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶JarInflaterCanvasクラス

JarInflaterCanvasクラスは、キャンバスとなるクラスです。

Star JarInflaterCanvas.java

```
import com.docomostar.StarApplicationManager; // ⓐ
import com.docomostar.io.HttpConnection;       //
import com.docomostar.media.MediaImage;        //
import com.docomostar.media.MediaManager;      //
import com.docomostar.ui.Canvas;               //
import com.docomostar.ui.Dialog;               //
import com.docomostar.ui.Graphics;             //
import com.docomostar.ui.Image;                //
import com.docomostar.util.JarInflater;        //
import javax.microedition.io.Connector;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

// JARファイルからのデータ読み込み(キャンバス)
public class JarInflaterCanvas extends Canvas
    implements Runnable {

    // 処理
    public void run() {
        // グラフィック
        Graphics g = getGraphics();

        JarInflater ji = null;
        try {
            // JARファイルとの接続
            ji = openJar(
                StarApplicationManager.getSourceURL()+  // ⓑ
                "test.jar");

            // JARファイルからのデータ読み込み
            String text  = readJarText(ji,"test.txt");
            Image  image = readJarImage(ji,"jyagi.gif");

            // JARファイルとの切断 …❹
            ji.close();

            // 描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
```

Next page.

```
            g.drawString("テキスト：",0,24*1);    // ⓒ
            g.drawString(text,0,24*2);           //
            g.drawString("イメージ：",0,24*4);        //
            g.drawImage(image,0,24*5);           //
            g.unlock(true);
        } catch (Exception e) {
            if (ji != null) ji.close();
            System.out.println("flag"+e.toString());
            showDialog("通信失敗");
        }
    }


    // JARファイルとの接続
    private JarInflater openJar(String url) throws Exception {
        HttpConnection c   = null;
        InputStream    in  = null;
        OutputStream   out = null;
        JarInflater    ji  = null;
        try {
            // JARファイルのサイズ取得
            in = Connector.openInputStream("scratchpad:///0");
            int jarSize = in.read()*256+in.read();
            in.close();

            // ネットからのJARファイルの読み込み …❶
            if (jarSize==0) {
                // 接続
                c = (HttpConnection)Connector.open(url,Connector.READ,true);
                c.setRequestMethod(HttpConnection.GET);
                c.connect();
                in = c.openInputStream();
                out = new ByteArrayOutputStream();

                // 読み込み
                byte[] w = new byte[10240];
                while (true) {
                    int size = in.read(w);
                    if (size<=0) break;
                    out.write(w,0,size);
                }

                // 切断
                out.close();
                in.close();
                c.close();

                // バイトデータ
                byte[] data = ((ByteArrayOutputStream)out).toByteArray();
```

```
                jarSize = data.length;

                // スクラッチパッドへの JAR ファイルの書き込み
                out = Connector.openOutputStream("scratchpad:///0;pos=0");
                out.write(jarSize/256);
                out.write(jarSize%256);
                out.write(data);
                out.close();
            }

            // JAR ファイルと接続 ...❷
            in = Connector.openInputStream("scratchpad:///0;pos=2,length="+jarSize);
            ji = new JarInflater(in);
            in.close();
            return ji;
        } catch (Exception e) {
            // 例外処理
            try {
                if (c  != null) c.close();
                if (in != null) in.close();
                if (out!= null) out.close();
                if (ji != null) ji.close();
            } catch (Exception e2) {
            }
            throw e;
        }
    }

    // JAR ファイルからのデータ読み込み ...❸
    private byte[] readJarData(JarInflater ji,String name) throws Exception {
        InputStream           in  = null;
        ByteArrayOutputStream out = null;
        try {
            in = ji.getInputStream(name);
            out = new ByteArrayOutputStream();
            byte[] w = new byte[10240];
            while (true) {
                int size = in.read(w);
                if (size<=0) break;
                out.write(w);
            }
            in.close();
            out.close();
            return out.toByteArray();
        } catch (Exception e) {
            try {
                in.close();
                out.close();
            } catch (Exception e2) {
```

Next page. ⬇

chapter
1
2
3
4
5
6
7
8
a

chapter

1

2

3

4

5

6

7

8

a

```
            }
            throw e;
        }
    }

    // JAR ファイルからのテキストの読み込み
    private String readJarText(JarInflater ji,String name) throws Exception {
        byte[] w = readJarData(ji,name);
        return new String(w,0,w.length);
    }

    // JAR ファイルからのイメージの読み込み
    private Image readJarImage(JarInflater ji,String name) throws Exception {
        byte[] w = readJarData(ji,name);
        MediaImage m = MediaManager.getImage(w);
        m.use();
        return m.getImage();
    }

    // ダイアログの表示
    private void showDialog(String text) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー");
        dialog.setText(text);
        dialog.show();
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa JarInflaterCanvas.java

ⓐ

```
import com.nttdocomo.io.HttpConnection;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.MediaManager;
import com.nttdocomo.util.JarInflater;
```

ⓑ

```
                IApplication.getCurrentApp().getSourceURL()+
```

ⓒ

```
            g.drawString("テキスト：",0,12*1);
            g.drawString(text,0,12*2);
            g.drawString("イメージ：",0,12*4);
            g.drawImage(image,0,12*5);
```

JarInflaterCanvas：書き込むバイトデータのフォーマット

スクラッチパッドにJARファイルを保存します。今回は書き込むバイトデータのフォーマットを次のように決めました。

0～1バイト目	JARファイルのバイトサイズ（0～20480）
2～20481バイト目	JARファイルのバイトデータ

先頭2バイトにJARファイルのバイトサイズ、それ以降にJARファイルのバイトデータを書き込みます。1バイトでは0～255しか表現できませんが、2バイトなら0～65535まで表現できます。

JarInflaterCanvas ...❶：ネットからJARファイルを読み込む

スクラッチパッドからJARファイルを読み込み処理を行い、サイズが0の時はまだJARファイルを保存していないと判断して、ネットからJARファイルを読み込みます。

JarInflaterCanvas ...❷：JARファイルと接続

スクラッチパッドに接続して入力ストリームを取得します。JARファイルをスクラッチパッドから読み込む時は「,length=バイトサイズ」の書式でバイトデータサイズを指定する必要があります。JARファイル以外の場合でも、この指定によって処理速度やメモリの負荷を軽減することができます。

JARファイルと接続するには、JarInflaterクラスを使います。コンストラクタに長さ指定付きの入力ストリームを渡してください。JarInflaterオブジェクトが生成できたら、入力ストリームをすぐに切断します。

［JarInflaterクラス］

JarInflater(*in*)

［解説］ JarInflaterクラスのコンストラクタ
［引数］ *in* 　長さ指定付きの入力ストリーム：InputStream型

JarInflaterCanvas …❸：JARファイルからのデータ読み込み

JARファイルからバイトデータを読み込むには、JarInflaterクラスのgetInputStream()メソッドを使います。

［JarInflaterクラス］

InputStream getInputStream(*name*)

［解説］ 入力ストリームの取得
［引数］ *name* ファイル名：String型
［戻り値］ 入力ストリーム

入力ストリームを取得できるので、それをテキストやイメージに変換します。

JarInflaterCanvas：JARファイルとの切断

JARファイルと切断するには、close()メソッドを使います。

［JarInflaterクラス］

void close()

［解説］ 切断を行う

▶ADFとエミュレータの設定

ADFの「UseNetwork」と「SPsize」を次のように設定してください。

UseNetwork	http
SPsize	20482

5-5 FeliCaアドホック通信



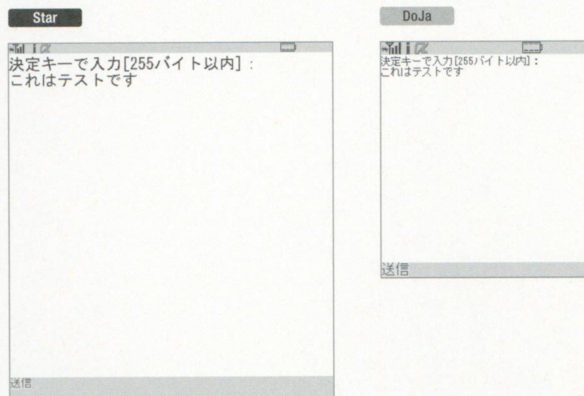
Star
http://book.mycom.co.jp/support/pc/star/55s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/55d.tml

FeliCaアドホック通信で端末間で文字列を送受信するプログラムを作ります。

Star　DoJa



FeliCaは非接触ICカードの技術で、読み取り端末にかざすだけで料金の精算などのデータのやりとりを行うことができます。電子マネー、電車の乗車券、社員証などに使われています。

FeliCaにはFeliCaチップを搭載した端末間の近距離通信を行うためのアドホック機能も搭載されており、端末に付いているFeliCaマークを向かい合わせることで、データをやりとりすることができます。受信側は通常の待ち受け状態であれば、受信データを感知した時に自動的に対応iアプリを起動して、受信し始めることができます。

このプログラムは、次の2つのクラスで構成されています。

- **FelicaObexEx**クラス（**FelicaObexEx.java**）
- **FelicaObexCanvas**クラス（**FelicaObexCanvas.java**）

プロジェクト名「**FelicaObexEx**」でプロジェクトを作成してください。

またFeliCaアドホック通信の機能はオプションAPIのため、利用できるかどうかは機種依存です。対応端末については以下のドキュメントを参照してください。

・iアプリコンテンツ開発ガイド for DoJa-5.x 各機種オプションAPI・拡張API実装状況
http://www.nttdocomo.co.jp/service/imode/make/content/iappli/technical_data/doja/

▶FelicaObexExクラス

FelicaObexExクラスは、プログラムの本体となるクラスです。

Star FelicaObexEx.java

```
import com.docomostar.StarApplication;
import com.docomostar.StarEventListener;
import com.docomostar.StarEventObject;
import com.docomostar.FelicaAdhocEvent;
import com.docomostar.ui.Display;
import java.util.Hashtable;

// FeliCa アドホック通信 ( 本体 )
public class FelicaObexEx extends StarApplication
    implements StarEventListener,Runnable { // …❶
    private FelicaObexCanvas c;

    // アプリの開始
    public void started(int launchType) {
        // アドホック通信の通知 …❶-1
        addEventListener(StarEventObject.STAR_FELICA_ADHOC_REQUEST_RECEIVED,this);

        // キャンバスの生成
        c = new FelicaObexCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }

    // アドホック通信の通知 …❶-2
    public void updateStarApplication(StarEventObject event) {
        switch(event.getType()){
        case StarEventObject.STAR_FELICA_ADHOC_REQUEST_RECEIVED:
            FelicaAdhocEvent evt = (FelicaAdhocEvent)event;
            requestReceived(evt.getReceivedParameter());
         }
    }

    // アドホック通信の処理 …❷
    public boolean requestReceived(Hashtable params) {
        Thread thread = new Thread(this);
        thread.start();
        return true;
    }

    // スレッドの処理
    public void run() {
        c.receiveAdHoc();
```

```
    }
}
```

DoJa FelicaObexEx.java

```
import com.nttdocomo.device.felica.FelicaAdhocListener;
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
import java.util.Hashtable;

// FeliCa アドホック通信 ( 本体 )
public class FelicaObexEx extends IApplication
    implements FelicaAdhocListener,Runnable { // …❶
    private FelicaObexCanvas c;

    // アプリの開始
    public void start() {
        c = new FelicaObexCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }

    // アドホック通信の処理 …❷
    public boolean requestReceived(Hashtable params) {
        Thread thread = new Thread(this);
        thread.start();
        return true;
    }

    // スレッドの処理
    public void run() {
        c.receiveAdHoc();
    }
}
```

FelicaObexEx ...❶：アドホック通信の通知

Star

FeliCaアドホック通信を行うiアプリは、他の端末からの通信を感知するためにイベントリスナーを追加する必要があります。StarApplicationクラスのaddEventListener()メソッドを呼び、イベント種別StarEventObject.STAR_FELICA_ADHOC_REQUEST_RECEIVEDとメソッドの実装先を指定します（❶-1)。

Star ［StarApplicationクラス］

void addEventListener(*event, listener*)

［解説］ イベントリスナーの追加
［引数］ *event* イベント：int型
listener リスナー：StarEventListener型

これによってFeliCaアドホック通信を感知した時、StarApplicationクラスのupdateStarApplication()メソッドが呼ばれるようになります（❶-2)。

Star ［StarApplicationクラス］

void updateStarApplication(*event*)

［解説］ イベント発生時に呼ばれる
［引数］ *event* イベント：StarEventObject型

このメソッドはFeliCaアドホック通信以外のイベント時にも呼ばれるので、event.getType()メソッドの値がStarEventObject.STAR_FELICA_ADHOC_REQUEST_RECEIVEDの時のみ処理を行うようにします。

引数*event*（イベント）をFelicaAdhocEventクラスにキャストし、getReceivedParameter()メソッドを呼ぶことで、送信元から送られてきたパラメータを取得することもできます。

Star ［FelicaAdhocEventクラス］

Hashtable getReceivedParameter()

［解説］ パラメータの取得
［戻り値］ パラメータ

受信後の処理はrequestReceived()メソッドで行っています。

DoJa

FeliCaアドホック通信を行うアプリは、iアプリの本体にFelicaAdhocListenerインタフェースを実装する必要があります。FelicaAdhocListenerインタフェースにはrequestReceived()メソッドが必要です。受信後の処理はこのrequestReceived()メソッドで行います。

FelicaObexEx ...❷：アドホック通信の処理

FeliCaアドホック通信の処理はrequestReceived()メソッドで行います。

DoJa ［FelicaAdhocListenerインタフェース］

boolean requestReceived(*receivedParams*)

［解説］ アドホック通信の受信
［引数］ *receivedParams* パラメータ：Hashtable型
［戻り値］ リクエストを承認

他の端末からのFeliCaアドホック通信を感知した時に呼ばれます。パラメータに応じてリクエストを承認するかどうかを返します。受信後の送信元との通信は、別スレッドを生成してそこで行います。そうしないと送信元の端末に承認したことが伝わらず、通信開始できなくなります。

▶FelicaObexCanvasクラス

FelicaObexCanvasクラスは、キャンバスとなるクラスです。

Star FelicaObexCanvas.java

```
import com.docomostar.device.felica.AdhocDataTransfer; // ⓐ
import com.docomostar.device.felica.Felica;           //
import com.docomostar.io.FelicaClientObexConnection;  //
import com.docomostar.io.ObexConnection;              //
import com.docomostar.io.ServerObexConnection;        //
import com.docomostar.ui.Canvas;                      //
import com.docomostar.ui.Display;                     //
import com.docomostar.ui.Dialog;                      //
import com.docomostar.ui.Graphics;                    //
import com.docomostar.ui.TextBox;                     //
import javax.microedition.io.Connector;
import java.io.InputStream;
import java.io.OutputStream;
```

Next page.

chapter

1

2

3

4

5

6

7

8

a

```
// FeliCa アドホック通信 ( キャンバス )
public class FelicaObexCanvas extends Canvas
    implements Runnable {
    private int    keyEvent = -999; // キーイベント
    private String text     = "";   // テキスト

    // 処理

 public void run() {
        // グラフィックス
        Graphics g = getGraphics();

        // ソフトラベル
        setSoftLabel(SOFT_KEY_1," 送信 ");

        while (true) {
            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString(" 決定キーで入力 [255 バイト以内 ] : ",0,24*1); // ⓑ
            g.drawString(text,0,24*2);                                //
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_SELECT) {
                // 文字入力
                imeOn(text,TextBox.DISPLAY_ANY,TextBox.KANA);
            } else if (keyEvent==Display.KEY_SOFT1) {
                // アドホック送信
                sendAdHoc(text.getBytes());
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // クライアント側の処理 ...❶
    private void sendAdHoc(byte[] data) {
        FelicaClientObexConnection c        = null;
        OutputStream               out      = null;
        AdhocDataTransfer          transfer = null;
```

```
        try {
            // FeliCaの接続 …❶_1
            Felica.open();
            transfer = Felica.getAdhocDataTransfer();
            transfer.setup("http://npaka.net/star10/star_dl/
                                               FelicaObexEx.jam","demo",null);

            // OBEXの接続 …❶_2
            c = (FelicaClientObexConnection)Connector.open(
                "obex:/felicaclient",Connector.WRITE,true);
            c.connect();
            c.setOperation(ObexConnection.PUT);
            c.setName("key");

            // データの読み書き …❶_3
            out = c.openOutputStream();
            out.write(data);
            out.close();
            c.sendRequest();
            if (c.getResponseCode() != ObexConnection.SUCCESS) {
                throw new Exception();
            }

            // OBEXの切断 …❶_4
            c.close();

            // FeliCaの切断 …❶_5
            transfer.terminateAdhoc();
            Felica.turnOffRFPower();
            Felica.close();
        } catch (Exception e) {
            try {
                if (out != null) out.close();
                if (c   != null) c.close();
                if (transfer != null) transfer.terminateAdhoc();
                Felica.turnOffRFPower();
                Felica.close();
            } catch (Exception e2) {
            }
        }
    }

    // サーバ側の処理 …❷
    public void receiveAdHoc() {
        ServerObexConnection c  = null;
        InputStream          in = null;
```

Next page. ⬇

```
        try {
            // OBEX の接続 ...❷_1
            c = (ServerObexConnection)Connector.open(
                "obex:/felicaserver",Connector.READ,true);
            c.accept();

            while (true) {
                c.receiveRequest();
                int op = c.getOperation();
                if (op==ObexConnection.PUT) {
                    if (c.getName().equals("key")) {
                        // データの読み書き ...❷_2
                        in = c.openInputStream();
                        byte[] w = new byte[c.getContentLength()];
                        in.read(w);
                        in.close();
                        text = new String(w);
                        c.sendResponse(ObexConnection.SUCCESS);
                    }
                }
                else if (op==ObexConnection.DISCONNECT) {
                    // OBEX の切断 ...❷_3
                    c.close();
                    break;
                }
            }
        } catch (Exception e) {
            try {
                if (in != null) in.close();
                if (c  != null) c.close();
            } catch (Exception e2) {
            }
        }
    }

    // ダイアログの表示
    private void showDialog(String text) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,"エラー");
        dialog.setText(text);
        dialog.show();
    }

    // IME イベントの処理
    public void processIMEEvent(int type,String text) {
        if (type==IME_COMMITTED) {
            if (text.getBytes().length <= 255) {
                this.text = text;
            } else {
                showDialog("255 バイト以内にして下さい");
```

```
            }
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```


DoJa FelicaObexCanvas.java

ⓐ

```
import com.nttdocomo.device.felica.AdhocDataTransfer;
import com.nttdocomo.device.felica.Felica;
import com.nttdocomo.io.FelicaClientObexConnection;
import com.nttdocomo.io.ObexConnection;
import com.nttdocomo.io.ServerObexConnection;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Dialog;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.TextBox;
```

ⓑ

```
            g.drawString("決定キーで入力[255バイト以内]：",0,12*1);
            g.drawString(text,0,12*2);
```

FelicaObex ...❶：クライアント側の処理

FeliCaアドホック通信は、クライアント側のiアプリからサーバ側iアプリに接続要求を出し、接続後にデータのやりとりを行います。そして最後に切断します。

FeliCaアドホック通信のクライアント側の処理は、次のような流れで行います。

❶_1：FeliCaの接続
❶_2：OBEXの接続
❶_3：データ読み書き
❶_4：OBEXの切断
❶_5：FeliCaの切断

FelicaObex ...❶_1：FeliCaの接続

FeliCaの接続を行うには、Felicaクラスのopen()メソッドを呼んだ後、AdhocDataTransferオブジェクトを取得し、setup()メソッドでアドホック通信による連続データ転送の接続を行います。AdhocDataTransferオブジェクトはアドホック通信の制御を行うためのクラスです。

［Felicaクラス］

static void open()

［解説］ FeliCaチップのオープン

［Felicaクラス］

static AdhocDataTransfer getAdhocDataTransfer()

［解説］ AdhocDataTransferオブジェクトの取得
［戻り値］ AdhocDataTransferオブジェクト

［AdhocDataTransferクラス］

void setup(*adfURL, command, params*)

［解説］ アドホック通信による連続データ転送の接続
［引数］ *adfURL* サーバ側iアプリのADFのURL：String型
command 起動コマンド：String型
params 起動パラメータ：Hashtable型

引数*adfURL*サーバ側iアプリのADFのURLは絶対パスで指定します。起動コマンドは任意の文字列を指定しますが、サーバ側iアプリのADFのAllowPushに「icc:起動コマンド」という文字列を指定する必要があります。今回は起動コマンドを「demo」としたので、「icc:demo」と記述します。

FelicaObex ...❶_2：OBEXの接続

OBEX（infrared object exchange protocol）とはデータのやりとりを行うためのプロトコルです。OBEXの接続を行うには、Connectorクラスのopen()メソッドを使います。

［Connectorクラス］

static Connection open(*name, mode, timeouts*)

［解説］ 接続を行う
［引数］ *name* 名前：String型
mode オープンモード：int型
timeouts タイムアウトを要求するかどうか：boolean型

引数の*mode*（オープンモード）には次の定数を指定します。

Connector.READ	読み込み
Connector.WRITE	書き込み
Connector.READ_WRITE	読み書き

クライアント側はopen()メソッドの第1引数に「obex:/felicaclient」を指定します。第2引数には、文字列の送信側なのでConnector.WRITEを指定します。

```
c = (FelicaClientObexConnection)Connector.open(
    "obex:/felicaclient",Connector.WRITE,true);
```

ConnectionオブジェクトをFelicaClientObexConnectionクラスにキャストして、connect()メソッドを呼び接続させます。そしてsetOperation()メソッドでリクエストの種類、setName()メソッドで名前を指定します。

［FelicaClientObexConnectionクラス］

void setOperation(*operation*)

［解説］ リクエストの種類の指定
［引数］ *operation*　リクエストの種類：int型

［FelicaClientObexConnectionクラス］

void setName(*name*)

［解説］ 名前の指定
［引数］ *name*　名前：String型

引数*operation*（リクエストの種類）には次の定数を指定します。

ObexConnection.GET	GETオペレーション
ObexConnection.PUT	PUTオペレーション
ObexConnection.DISCONNECT	DISCONNECTオペレーション

FelicaObex ...❶_3：データの読み書き

FelicaClientObexConnectionクラスのopenOutputStream()メソッドで出力ストリームが取得できるので、そこに書き込みます。

［FelicaClientObexConnectionクラス］

OutputStream openOutputStream()

［解説］ 出力ストリームの取得
［戻り値］ 出力ストリーム

その後sendRequest()メソッドでリクエストを送信します。

［FelicaClientObexConnectionクラス］

void sendRequest()

［解説］ リクエストを送信してレスポンスの受信を完了するまで待つ

最後にgetResponseCode()メソッドの返り値を調べ、ObexConnection.SUCCESSかどうかで、通信が成功したかどうかが分かります。

［FelicaClientObexConnectionクラス］

int getResponseCode()

［解説］ レスポンスコードの取得
［戻り値］ レスポンスコード

FelicaObex …❶_4：OBEXの切断

OBEXの切断を行うには、FelicaClientObexConnectionクラスのclose()メソッドを使います。

［FelicaClientObexConnectionクラス］

```
void close()
```

［解説］ OBEXの切断

FelicaObex …❶_5：FeliCaの切断

FeliCaを切断にするには、AdhocDataTransferオブジェクトのterminateAdhoc()メソッドと、FelicaクラスのturnOffRFPower()メソッドとclose()メソッドを呼びます。

［AdhocDataTransferクラス］

```
void terminateAdhoc()
```

［解説］ アドホック通信による連続データ転送の終了

［Felicaクラス］

```
static void turnOffRFPower()
```

［解説］ 搬送波の出力の停止

［Felicaクラス］

```
static void close()
```

［解説］ FeliCaチップのクローズ

FelicaObex ...❷：サーバ側の処理

FeliCaアドホック通信の受信側の処理は、次のような流れで行います。

❷_1：OBEXの接続
❷_2：データ読み書き
❷_3：OBEXの切断

FelicaObex ...❷_1：OBEXの接続

OBEXの接続を行うには、Connectorクラスのopen()メソッドを使います。

サーバ側はopen()メソッドの第1引数に「obex:/felicaserver」を指定します。第2引数には文字列の受信側なのでConnector.READを指定します。

［FelicaObex.Javaの一部］

```
c = (ServerObexConnection)Connector.open(
    "obex:/felicaserver",Connector.READ,true);
```

ConnectionオブジェクトをServerObexConnectionインタフェースにキャストして、accept()メソッドを呼び、クライアント側からの接続要求が来るまで待ちます。

接続要求がきたらgetOperation()メソッドでリクエストの種類、getName()メソッドで名前を取得し、それに応じて各種処理を行います。

［ServerObexConnectionインタフェース］

int getOperation()

［解説］　リクエストの種類の取得
［戻り値］　リクエストの種類

［ServerObexConnectionインタフェース］

String getName()

［解説］　名前の取得
［戻り値］　名前

FelicaObex ...❷_2：データの読み書き

ServerObexConnectionクラスのopenInputStream()メソッドで入力ストリームが取得できるので、そこから読み込みます。データの長さはgetContentLength()メソッドで取得します。

［ServerObexConnectionインタフェース］

InputStream openInputStream()

［解説］ 入力ストリームの取得
［戻り値］ 入力ストリーム

［ServerObexConnectionインタフェース］

int getContentLength()

［解説］ データの長さの取得
［戻り値］ データの長さ

その後sendResponse()メソッドでレスポンスを返します。

［ServerObexConnectionインタフェース］

void sendResponse(*response*)

［解説］ レスポンスを返す
［引数］ *response* レスポンスコード：int型

FelicaObex ...❷_3：OBEXの切断

最後にクライアント側と同様にOBEXの切断を行います。

▶ADFの設定

ADFの「PackageURL」「AllowPushBy」「LaunchByApp」「LaunchApp」を次のように設定してください。

PackageURL	http://npaka.net/star10/star_dl/FelicaObexEx.jar
AllowPushBy	icc:demo
LaunchByApp	deny
LaunchApp	yes

「PackageURL」はエミュレータでも実行できるように絶対パスを指定します。

「AllowPushBy」には「icc:起動コマンド」を指定します。クライアント側の時はiアプリ起動する側、サーバ側の時はiアプリを起動される側なので、「LaunchByApp」と「LaunchApp」の両方を有効にしています。

3Dグラフィックス

chapter
1
2
3
4
5
6
7
8
a

本章では、iアプリで3Dグラフィックスを表示する方法について説明します。iアプリの3Dグラフィックスエンジンには「マスコットカプセル」と「OpenGL ES」の2種類がありますが、OpenGL ESはiアプリDXの機能のため勝手アプリでは利用できないので、マスコットカプセルについて解説します。

6-1 3Dモデルの表示



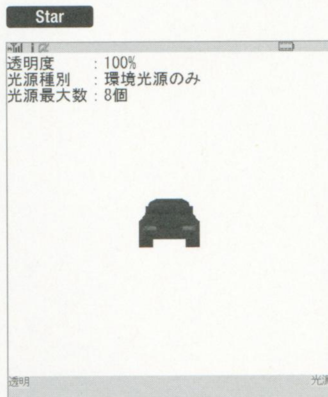
Star
http://book.mycom.co.jp/support/pc/star/61s.html



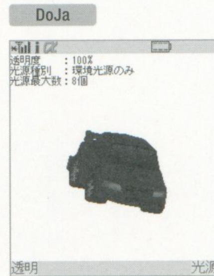
DoJa
http://book.mycom.co.jp/support/pc/star/d/61d.html

3Dモデルを表示するプログラムを作ります。

Star



DoJa



方向キー	3Dモデルの回転
ソフトキー 1	透明度の変更
ソフトキー 2	光源種類の変更

このプログラムは、次の2つのクラスで構成されています。

- **Graphics3DExクラス（Graphics3DEx.java）**
- **GraphicsCanvasクラス（GraphicsCanvas.java）**

プロジェクト名「**Graphics3DEx**」でプロジェクトを作成してください。

▶3Dデータの準備

今回使用する3Dデータは次の1つです。プロジェクトの「res」フォルダに置いてください。

- 3Dデータ：**car.d4d**

iアプリで利用する3Dモデルのファイル形式は**マスコットカプセル**ver.4用の3Dデータ**D4Dファイル**です。マスコットカプセルはエイチアイが開発した3Dグラフィックスエンジンで、NTTドコモ、ソフトバンク、auをはじめとする多くの携帯端末で採用されています。

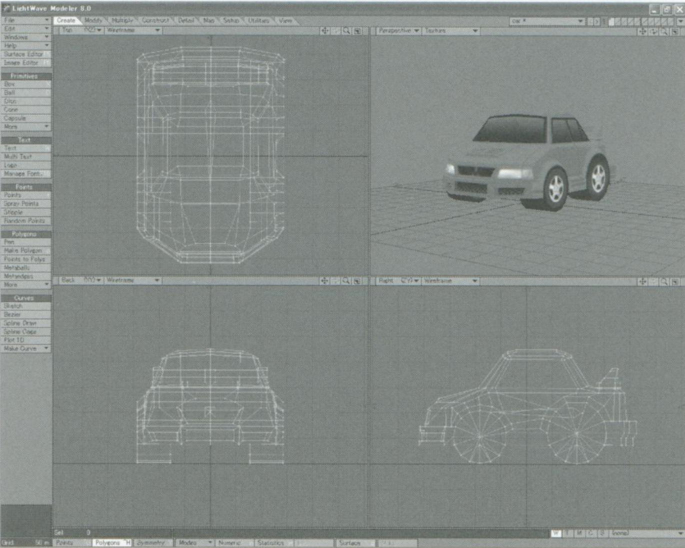
Column» 3Dデータ変換ツールの準備

D4Dファイルの作成に必要な開発ツールは次の3つです。

3DCG作成ソフト

エイチアイ社よりデータ出力用のプラグインが提供されている3DCG作成ソフトは次の6種類です。

3ds Max	8.0/9.0/2008/2009（Windows 32bit）
Maya	5.0/6.0/6.5/7.0/8.0/8.5/2008（Windows 32bit）
LightWave3D	7.5 or later（Windows 32bit）
SOFTIMAGE|3D	3.9.22/4.0（Windows 32bit）
SOFTIMAGE|XSI	5.11/6.x/7.0（Windows 32bit）
Blender	2.41



H3T Exporter

H3T Exporterはエイチアイより提供されている**H3Tファイル**出力用のプラグインで、

以下のサイトより入手できます。ドキュメントの指示に従って設定を行ってください。

・MascotCapsule Toolkit for DoCoMo
http://www.mascotcapsule.com/toolkit/docomo/ja/index.php

D4D Tool Kit とプラグイン

D4D Tool Kitは3DCG作成ソフトからプラグインで出力したデータ（*.h3t）を、d4dファイルに変換するためのツール群です。

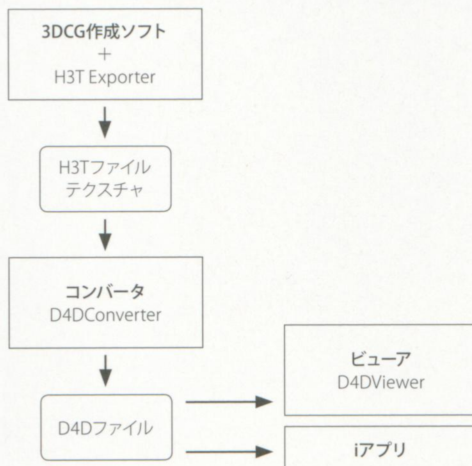
D4DConverter	H3TからD4Dへのコンバータ
D4DEditer	ver.3形式（mbac/mtra/bmp）からD4Dd形式へのコンバータ
D4DViewer	d4dファイルビューア

インストールパッケージはプラグインをダウンロードしたサイトで入手できるので、ダウンロード後インストールしてください。

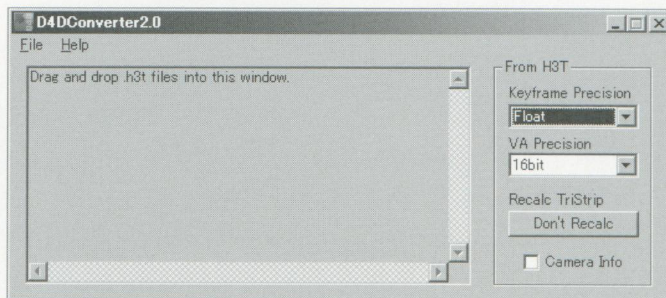
Column»　3Dデータの作成の流れ

市販の3DCG作成ソフトで作ったモデルを**D4Dファイル**に変換する方法について解説します。

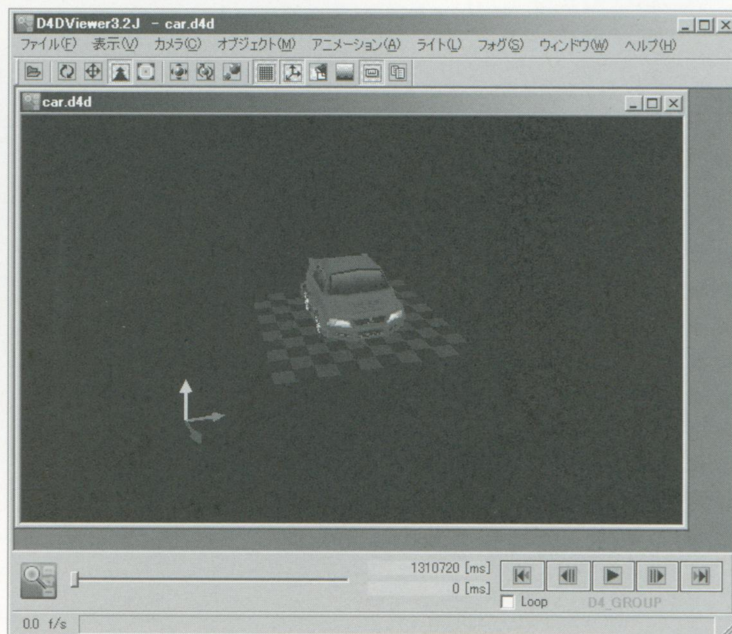
3DCG作成ソフトでモデルを作成し、H3T Exporter経由でH3Tファイルという中間データを出力します。そして、**D4DConverter**でD4Dファイルに変換します。最後にD4Dビューアで動作チェックを行います。



D4DConverterは、H3TファイルをD4Dファイルに変換するためのコンバータです。
D4DConverterを起動するとウィンドウが開きます。そのウィンドウにH3Tファイルをドラッグ＆ドロップすればD4Dファイルが生成されます。



D4DViewerはD4Dファイルが正しいデータかどうかを確認するためのビューアです。
D4DViewerを起動するとウィンドウが開きます。そこにD4Dファイルをドラッグ＆ドロップするとモデルが表示されます。緑の三角ボタンをクリックするとアクションが再生されます。



▶Graphics3DExクラス

Graphics3DExクラスは、プログラムの本体となるクラスです。

Star Graphics3DEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// 3D モデルの表示 ( 本体 )
public class Graphics3DEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {           // ⓒ
        Graphics3DCanvas c = new Graphics3DCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa Graphics3DEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class Graphics3DEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶Graphics3DCanvasクラス

Graphics3DCanvasクラスはキャンバスとなるクラスです。

Star Graphics3DCanvas.java

```
import com.docomostar.ui.graphics3d.Graphics3D;         // ⓐ
import com.docomostar.ui.graphics3d.ActionTable;        //
import com.docomostar.ui.graphics3d.DrawableObject3D;   //
import com.docomostar.ui.graphics3d.Figure;             //
import com.docomostar.ui.graphics3d.Group;              //
```

Next page. ⬇

```
import com.docomostar.ui.graphics3d.Light;                //
import com.docomostar.ui.graphics3d.Object3D;             //
import com.docomostar.ui.util3d.Transform;                //
import com.docomostar.ui.util3d.Vector3D;                 //
import com.docomostar.ui.Canvas;                          //
import com.docomostar.ui.Display;                         //
import com.docomostar.ui.Graphics;                        // ⓐ
import javax.microedition.io.Connector;
import java.io.InputStream;

// 3Dモデルの表示(キャンバス)
public class Graphics3DCanvas extends Canvas
    implements Runnable {
    // システム
    private Graphics   g = getGraphics();  // グラフィックス
    private Graphics3D g3 = (Graphics3D)g; // 3Dグラフィックス …❶
    private int        keyEvent = -999;    // キーイベント
    private int        keyState = 0;       // キー状態

    // 3Dオブジェクト
    private Group       group;  // グループ
    private Figure      figure; // モデル
    private ActionTable table;  // アクションテーブル
    private int         frame;  // フレーム

    // 座標
    private Transform trans = new Transform(); // 視点座標

    // 光源
    private int      lightType = 0;                    // 種別
    private int      lightMax = Light.getMaxLights();  // 最大数
    private String[] lightName = {                     // 名前
        "環境光源のみ",    "環境光源+平行光源",
        "環境光源+点光源","環境光源+スポット光源"};

    // アルファブレンディング
    private int transRate = 100; // 割合

    // 処理
    public void run() {
        // 3Dオブジェクトの読み込み …❷
        try {
            InputStream in = Connector.openInputStream(
                "resource:///car.d4d");                  //…❷-1
            group = (Group)Object3D.createInstance(in); //…❷-1
            in.close();
            figure = (Figure)group.getElement(0);
            table = (ActionTable)group.getElement(1);
        } catch (Exception e) {
```

```
        e.printStackTrace();
    }

    // テクスチャの歪み補正 ...❸
    figure.setPerspectiveCorrectionEnabled(true);

    // 視点座標の指定 ...❹
    trans.lookAt(
        new Vector3D(0,50,100), // 視点
        new Vector3D(0,50,0),   // 参照点
        new Vector3D(0,1,0));   // UP ベクトル

    // 変換行列の拡大縮小 ...❺
    trans.scale(20,20,20);

    // 光源の指定
    changeLight(0);

    // ソフトラベル
    setSoftLabel(SOFT_KEY_1," 透明 ");
    setSoftLabel(SOFT_KEY_2," 光源 ");
    while (true) {
        // 変換行列の回転 ...❻
        keyState = getKeypadState();
        if (((1<<Display.KEY_UP)&keyState) != 0) {
            trans.rotate(1,0,0,10);
        }
        if (((1<<Display.KEY_DOWN)&keyState) != 0) {
            trans.rotate(1,0,0,-10);
        }
        if (((1<<Display.KEY_LEFT)&keyState) != 0) {
            trans.rotate(0,1,0,-10);
        }
        if (((1<<Display.KEY_RIGHT)&keyState) != 0) {
            trans.rotate(0,1,0,10);
        }

        // アルファブレンディングの指定 ...⓬
        if (keyEvent==Display.KEY_SOFT1) {
            transRate += 10;
            if (transRate>100) transRate = 0;
            if (transRate==100) {
                figure.setBlendMode(DrawableObject3D.BLEND_NORMAL);
            } else {
                figure.setBlendMode(DrawableObject3D.BLEND_ALPHA);
            }
            figure.setTransparency(transRate);
        }
```

Next page. ⬇

```
        // 光源の指定
        if (keyEvent==Display.KEY_SOFT2) {
            if (++lightType>3) lightType = 0;
            changeLight(lightType);
        }
        keyEvent = -999;

        // アクションのフレーム指定 ...⓭
        frame += 65536;
        if (frame >= table.getMaxFrame(0)) frame = 0;
        figure.setTime(frame);

        // 画面の描画
        g.lock();
        g.setColor(g.getColorOfName(g.WHITE));
        g.fillRect(0,0,480,480);                                        // ⓑ
        g.setColor(g.getColorOfName(g.BLACK));                          //
        g.drawString("透明度    :"+transRate+"%",        0,24*1);  //
        g.drawString("光源種別  :"+lightName[lightType],0,24*2);  //
        g.drawString("光源最大数:"+lightMax+"個",       0,24*3);  //

        // モデルの描画 ...⓮
        g3.renderObject3D(group,trans);
        g3.flushBuffer();
        g.unlock(true);

        // スリープ
        try {
            Thread.sleep(100);
        } catch (Exception e) {
        }
    }
}

// 光源の指定 ...❼
private void changeLight(int type) {
    Light light;

    // 初期化
    g3.resetLights();

    // 環境光源 ...❽
    light = new Light();
    light.setMode(Light.AMBIENT);        // 種類
    light.setColor(255<<16|255<<8|255); // 色
    light.setIntensity(0.5f);            // 強度
    g3.addLight(light,null);             // 追加
```

```
        // 平行光源 ...❾
        if (type==1) {
            light = new Light();
            light.setMode(Light.DIRECTIONAL);       // 種類
            light.setIntensity(1);                  // 強度
            light.setVector(new Vector3D(-1,1,0)); // 向き（左下）
            g3.addLight(light,null);                // 追加
        }

        // 点光源 ...❿
        if (type==2) {
            light = new Light();
            light.setMode(Light.OMNI);                        // 種類
            light.setColor(255<<16|255<<8|255);               // 色
            light.setIntensity(1);                            // 強度
            light.setAttenuation(1,0,0);                      // 減衰
            light.setPosition(new Vector3D(100,-100,0)); // 位置（右上）
            g3.addLight(light,null);                          // 追加
        }

        // スポット光源 ...⓫
        if (type==3) {
            light = new Light();
            light.setMode(Light.SPOT);                 // 種類
            light.setColor(255<<16|255<<8|255);        // 色
            light.setIntensity(1);                     // 強度
            light.setAttenuation(1,0,0);               // 減衰
            light.setSpotAngle(45);                    // スポット光源の角度
            light.setSpotExponent(0);                  // スポット光源の輝度分布
            light.setPosition(new Vector3D(100,-100,0)); // 位置（右上）
            light.setVector(new Vector3D(-1,1,0));       // 向き（左下）
            g3.addLight(light,null);                     // 追加
        }
    }

    // キーイベントの処理
    public void processEvent(int type,int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

chapter

1

2

3

4

5

6

7

8

a

DoJa Graphics3DCanvas.java

ⓐ

```
import com.nttdocomo.ui.graphics3d.ActionTable;
import com.nttdocomo.ui.graphics3d.DrawableObject3D;
import com.nttdocomo.ui.graphics3d.Figure;
import com.nttdocomo.ui.graphics3d.Graphics3D;
import com.nttdocomo.ui.graphics3d.Group;
import com.nttdocomo.ui.graphics3d.Light;
import com.nttdocomo.ui.graphics3d.Object3D;
import com.nttdocomo.ui.util3d.Transform;
import com.nttdocomo.ui.util3d.Vector3D;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
```

ⓑ

```
            g.fillRect(0,0,240,240);
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("透明度    :"+transRate+"%",       0,12*1);
            g.drawString("光源種別  :"+lightName[lightType],0,12*2);
            g.drawString("光源最大数:"+lightMax+"個",       0,12*3);
```

Graphics3DCanvas ...❶:Graphics3Dオブジェクトの準備

2Dグラフィックスの描画にGraphicsクラスを利用するのに対し、3Dグラフィックスの描画に**Graphics3Dインタフェース**を利用します。Graphics3DインタフェースのオブジェクトはGraphicsオブジェクトをキャストすることにより取得します。

Graphics3DCanvas ...❷:3Dオブジェクトの読み込み

3Dデータ「car.d4d」を読み込むには、openInputStream()メソッドを使い、入力ストリームオブジェクトとして取得します(❷-1)。3Dオブジェクトを生成するにはObject3DクラスのcreateInstance()メソッドを使います(❷-2)。

[Object3Dクラス]

static Object3D createInstance(*data*)

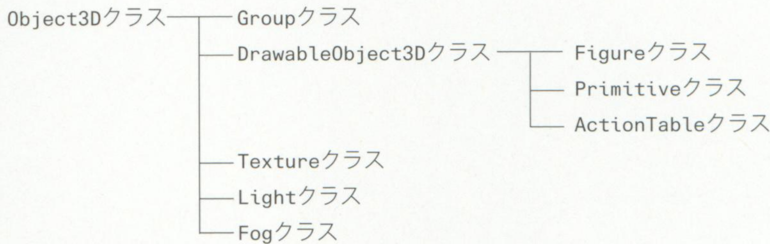
[解説] 3Dオブジェクトの生成
[引数] *data* バイトデータ:byte[]型
[戻り値] 3Dオブジェクト

［Object3Dクラス］

static Object3D createInstance(*in*)

［解説］ 3Dオブジェクトの生成
［引数］ *in* 入力ストリーム：InputStream型
［戻り値］ 3Dオブジェクト

3DオブジェクトとはObject3Dクラスを継承しているオブジェクトで、次の8つのクラスが存在します。



Groupオブジェクトに含まれている3Dオブジェクトを取得するには、**Groupクラス**のgetElement()メソッドを使います。

［Groupクラス］

Object3D getElement(*index*)

［解説］ 3Dオブジェクトの取得
［引数］ *index* 3Dオブジェクトのインデックス：int型
［戻り値］ 3Dオブジェクト

Graphics3DCanvas …❸：テクスチャの歪み補正

テクスチャの歪み補正の有効/無効を指定するには、**DrawableObject3Dクラス**のsetPerspectiveCorrectionEnabled()メソッドを使います。初期状態では無効になっています。Groupオブジェクトに対して行った時は、Groupオブジェクトが参照している全ての3Dオブジェクトの設定に反映されます。

［DrawableObject3Dクラス］

void setPerspectiveCorrectionEnabled(*flag*)

［解説］ テクスチャの歪み補正の有効化/無効化を指定
［引数］ *flag* 有効/無効：boolean型

Graphics3DCanvas ...❹：視点座標の指定

視点座標の指定を行うには、**Transformクラス**を使います。Transformクラスは変換行列の情報を保持するクラスで、回転・平行移動・拡大縮小という操作の他に、視点から参照点を見た時の変換行列を生成するlookAt()メソッドを持っています。

［Transformクラス］

void lookAt(*position, look, up*)

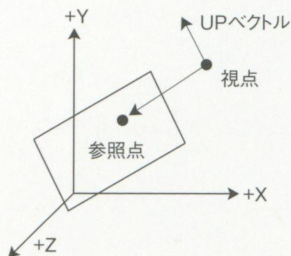
［解説］ 視点から参照点を見た時の変換行列の計算

［引数］ *position* 視点：Vector3D型

look 参照点：Vector3D型

up UPベクトル：Vector3D型

視線方向とUPベクトルが垂直でない時は、視線方向と垂直になるようにUPベクトルが修正された上で、変換行列が計算されます。



Graphics3DCanvas ...❺：変換行列の拡大縮小

変換行列の拡大縮小を行うには、Transformクラスのscale()メソッドを使います。

［Transformクラス］

void scale(*x, y, z*)

［解説］ 変換行列の拡大縮小

［引数］ *x* X軸方向の倍率：float型

Y Y軸方向の倍率：float型

Z Z軸方向の倍率：float型

［Transformクラス］

void scale(*v*)

［解説］ 変換行列の拡大縮小

［引数］ *v* XYZ軸方向の倍率：Vector3D型

Graphics3DCanvas …❻：変換行列の回転

変換行列の回転を行うには、Transformクラスのrotate()メソッドを使います。初期状態からどの程度回転させるかでなく、現在の角度からどの程度回転させるかを指定します。

［Transformクラス］

void rotate(*x, y, z, angle*)

［解説］ 変換行列の回転

［引数］ *x* 回転軸のXベクトル：float型
y 回転軸のYベクトル：float型
z 回転軸のZベクトル：float型
angle 角度（単位は度）：float型

［Transformクラス］

void rotate(*v, angle*)

［解説］ 変換行列の回転

［引数］ *v* 回転軸：Vector3D型
angle 角度（単位は度）：float型

Graphics3DCanvas ...❼：光源の指定

光源を追加するには**Lightクラス**のオブジェクトを生成し、Graphics3DインタフェースのaddLight()メソッドを使います。

［Graphics3Dインタフェース］

void addLight(*light, trans*)

［解説］ 光源の追加
［引数］ *light*　光源：Light型
　　　　trans　光源位置：Transform型

光源の種類は**環境光**、**平行光源**、**点光源**、**スポット光源**の4つがあり、LightクラスのsetMode()メソッドで指定します。

［Lightクラス］

void setMode(*mode*)

［解説］ 光源の種類の指定
［引数］ *mode*　光源の種類：int型

引数 *mode*（光源の種類）には次の定数を指定します。

Light.AMBIENT	環境光源
Light.DIRECTIONAL	平行光源
Light.OMNI	点光源
Light.SPOT	スポット光源

光源を解除するにはGraphics3DインタフェースのresetLights()メソッドを使います。

［Graphics3Dインタフェース］

void resetLights()

［解説］ 光源の解除

光源がいくつ指定できるかは機種依存で、LightクラスのgetMaxLights()メソッドで取得できます。

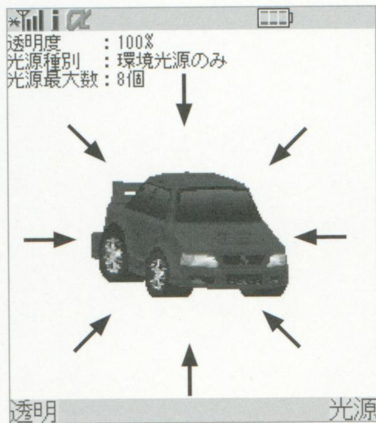
［Lightクラス］

int getMaxLights()

［解説］　光源の最大数を取得
［戻り値］ 光源の最大数

Graphics3DCanvas ...❽：環境光源

環境光源は、光源からの光が何度となく反射を繰り返して、方向を特定できない光のことで、どの位置でも一定に明るくします。



光源の色はLightクラスのsetColor()メソッド、光源の強度はLightクラスのsetIntensity()メソッドで指定します。

［Lightクラス］

void setColor(*color*)

［解説］ 光源の色の指定

［引数］ *color*　色（0x00RRGGBBの形式）：int型

［Lightクラス］

void setIntensity(*intensity*)

［解説］ 光源の強度の指定

［引数］ *intensity*　強度通常0～1の間で指定、1が強い光：float型

Graphics3DCanvas …❾：平行光源

平行光源は、空間中のあらゆる場所に同一方向から照らされる光の光源で、光の当たっている場所を明るくします。



平行光源とスポット光源の角度はLightクラスのsetVector()メソッドで指定します。

［Lightクラス］

void setVector(*v*)

［解説］ 光源の向きの指定

［引数］ *v*　光源の向き：Vector3D型

Graphics3DCanvas ...⑩：点光源

点光源は、光源の位置から放射線状に影響を及ぼす光源で、距離により減衰させることが可能です。



点光源とスポット光源の減衰はLightクラスのsetAttenuation()メソッドで指定します。

［Lightクラス］

void setAttenuation(*constant, linear, quadratic*)

［解説］　光源の減衰に関するパラメータの指定

［引数］　*constant*　　constant値：float型

linear　　linear値：float型

quadratic　　quadratic値：float型

引数の値と次の数式により、減衰する量が定義されます。

```
1 / (constant + linear * 頂点と光源の距離
    + quadratic * 頂点と光源の距離 * 頂点と光源の距離 )
```

点光源とスポット光源の位置はLightクラスのsetPosition()メソッドで指定します。

［Lightクラス］

void setPosition(v)

［解説］　光源の位置の指定

［引数］　*v*　　位置：Vector3D型

Graphics3DCanvas ...⓫：スポット光源

スポット光源は、光源の位置から方向ベクトルに向かってコーン（円すい）状に影響を及ぼす光源で、距離により減衰させることが可能です。



スポット光源の強度はLightクラスのsetSpotAngle()メソッドで指定します。

［Lightクラス］

void setSpotAngle(*angle*)

［解説］ スポット光源の角度の指定
［引数］ *angle* スポット光源の円すいの角度（単位は度）：float型

スポット光源の強度はLightクラスのsetSpotExponent()メソッドで指定します。

［Lightクラス］

void setSpotExponent(*exponent*)

［解説］ スポット光源モード用の輝度分布の指定
［引数］ *exponent* 輝度分布：float型

引数の*exponent*（輝度分布）は0の時は均一、128の時は濃淡がはっきりした明るさの分布になります。

Graphics3DCanvas …⑫：アルファブレンディングの指定

モデルの**ブレンドモード**を変更するには、**Figure**クラスのsetBlendMode()メソッドを使います。

［Figureクラス］

void setBlendMode(*mode*)

［解説］ ブレンドモードの指定

［引数］ *mode* ブレンドモード：int型

引数の*mode*（ブレンドモード）には次の定数を指定します。

DrawableObject3D.BLEND_NORMAL	通常（ブレンドなし）
DrawableObject3D.BLEND_ALPHA	アルファブレンド （背景 x（100% - 透明度）+ プリミティブ x 透明度）
DrawableObject3D.BLEND_ADD	加算 （背景 x 100% + プリミティブ x 透明度）

透明度を指定するには、FigureクラスのsetTransparency()メソッドを使います。

［Figureクラス］

void setTransparency(*rate*)

［解説］ 透明度をパーセントで設定

［引数］ *rate* 透明度（単位はパーセント）：float型

引数の*rate*（透明度）には0～100を指定します（100の時が不透明）。ブレンドモードがアルファブレンドか加算の時のみ、この設定が反映されます。

Graphics3DCanvas ...⑬：アクションのフレーム指定

モデルに何フレーム目のポーズをとらえるかを指定するには、FigureクラスのsetTime()メソッドを使います。

［Figureクラス］

void setTime(*frame*)

［解説］ アクションのフレーム指定
［引数］ *frame*　フレーム：int型

引数の *frame*（フレーム）を増やしながら表示することにより、モデルをアニメーションさせることができます。最大フレーム数は、**ActionTableクラス**のgetMaxFrame()メソッドで取得できます。

［ActionTableクラス］

int getMaxFrame(*index*)

［解説］　最大フレーム数の取得
［引数］　*index*　アクションインデックス：int型
［戻り値］　最大フレーム数

Graphics3DCanvas ...⑭：モデルの描画

モデルを描画するにはGraphics3DインタフェースのrenderObject3D()メソッドを使います。

［Graphics3Dインタフェース］

void renderObject3D(*obj3d, trans*)

［解説］ 3Dオブジェクトのレンダリング
［引数］ *obj3d*　3Dオブジェクト：Object3D型
　　　　trans　変換行列：Transform型

レンダリング結果を実際に描画に反映させるにはGraphics3DインタフェースのflushBuffer()メソッドを使います。

［Graphics3Dインタフェース］

void flushBuffer()

［解説］ レンダリング結果の画面への反映

6-2 プリミティブの表示

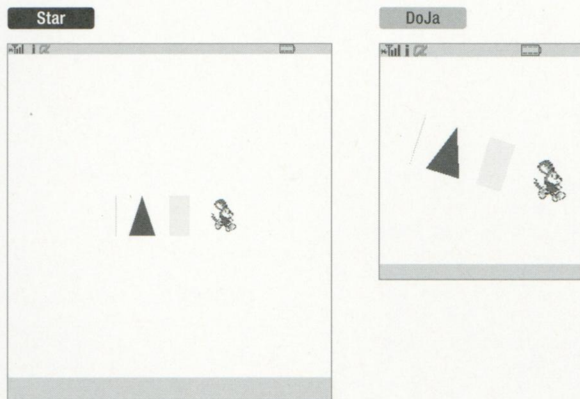


Star
http://book.mycom.co.jp/support/pc/star/62s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/62d.html

プリミティブを表示するプログラムを作ります。



方向キー	プリミティブの回転

プリミティブとは3Dグラフィックスを描画する時に基本になる単位図形のことで、

・点　・線　・三角形　・四角形　・ポイントスプライト

の5種類があります。ポイントスプライトは1頂点にテクスチャを張ったもので、常に視点の方向にイメージを表示します。

このプログラムは、次の2つのクラスで構成されています。

- **PrimitiveExクラス(PrimitiveEx.java)**
- **PrimitiveCanvasクラス(PrimitiveCanvas.java)**

プロジェクト名「**PrimitiveEx**」でプロジェクトを作成してください。

▶テクスチャの準備

今回使用するテクスチャは1つです。プロジェクトの「res」フォルダに置いてください。テクスチャとして使用できるイメージフォーマットは、256色で縦横256ドット以下の無圧縮BMPです。

・テクスチャファイル：**jyagi.bmp**（48x48ドット）



▶PrimitiveExクラス

PrimitiveExクラスは、プログラムの本体となるクラスです。

Star PrimitiveEx.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// プリミティブの表示 ( 本体 )
public class PrimitiveEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {          // ⓒ
        PrimitiveCanvas c = new PrimitiveCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa PrimitiveEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class PrimitiveEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶PrimitiveCanvasクラス

PrimitiveCanvasクラスはキャンバスとなるクラスです。

Star PrimitiveCanvas.java

```
import com.docomostar.ui.graphics3d.Graphics3D; // ⓐ
import com.docomostar.ui.graphics3d.Object3D;   //
import com.docomostar.ui.graphics3d.Primitive;  //
import com.docomostar.ui.graphics3d.Texture;    //
import com.docomostar.ui.util3d.Transform;      //
import com.docomostar.ui.util3d.Vector3D;       //
import com.docomostar.ui.Canvas;                //
import com.docomostar.ui.Display;               //
import com.docomostar.ui.Graphics;              //
import javax.microedition.io.Connector;         //
import java.io.InputStream;                     //

// プリミティブの表示（キャンバス）
public class PrimitiveCanvas extends Canvas
    implements Runnable {
    // システム
    private Graphics   g = getGraphics();  // グラフィックス
    private Graphics3D g3 = (Graphics3D)g; // 3D グラフィックス
    private int        keyState = 0;       // キー状態

    // プリミティブ
    private Primitive point;       // 点
    private Primitive line;        // ライン
    private Primitive triangle;    // 三角形
    private Primitive quad;        // 四角形
    private Primitive pointSprite; // ポイントスプライト
    private Texture   texture;     // テクスチャ

    // 座標
    private Transform trans = new Transform();  // 視点座標
    private Transform origin = new Transform(); // 原点

    // 処理
    public void run() {
        int[] arr;

        // 点
        point = new Primitive(
            Primitive.PRIMITIVE_POINTS,
            Primitive.COLOR_PER_PRIMITIVE,1); // …❶
        arr = point.getVertexArray();         // …❷
        arr[0]=-100; arr[1]=0; arr[2]=0;      // 頂点（-100,0,0）
```

Next page. ⬇

```
point.getColorArray()[0] = 255<<16;    // 色(赤) ...❸

// 線
line = new Primitive(
    Primitive.PRIMITIVE_LINES,
    Primitive.COLOR_PER_PRIMITIVE,1); // ...❶
arr = line.getVertexArray();          // ...❷
arr[0]=-80; arr[1]=-30; arr[2]=0;     // 頂点(-80,-30,0)
arr[3]=-80; arr[4]= 30; arr[5]=0;     // 頂点(-80, 30,0)
line.getColorArray()[0] = 255<<8;     // 色(緑) ...❸

// 三角形
triangle = new Primitive(
    Primitive.PRIMITIVE_TRIANGLES,
    Primitive.COLOR_PER_PRIMITIVE|
    Primitive.NORMAL_NONE,1);       // ...❶
arr = triangle.getVertexArray();    // ...❷
arr[0]=-40; arr[1]= 30; arr[2]=0;  // 頂点(-40,30,0)
arr[3]=-20; arr[4]=-30; arr[5]=0;  // 頂点(-20,-30,0)
arr[6]=-60; arr[7]=-30; arr[8]=0;  // 頂点(-60,-30,0)
triangle.getColorArray()[0] = 255; // 色(青) ...❸

// 四角形
quad = new Primitive(
    Primitive.PRIMITIVE_QUADS,
    Primitive.COLOR_PER_PRIMITIVE|
    Primitive.NORMAL_NONE,1);      // ...❶
arr = quad.getVertexArray();       // ...❷
arr[0]= 0; arr[1]=-30; arr[2]=0;  // 頂点(0,-30,0)
arr[3]=30; arr[4]=-30; arr[5]=0;  // 頂点(3,-30,0)
arr[6]=30; arr[7]= 30; arr[8]=0;  // 頂点(3, 30,0)
arr[9]= 0; arr[10]=30; arr[11]=0; // 頂点(0, 30,0)
quad.getColorArray()[0] = (255<<16)|(255<<8); // 色(黄)...❸

// ポイントスプライト
pointSprite = new Primitive(
    Primitive.PRIMITIVE_POINT_SPRITES,
    Primitive.POINT_SPRITE_PER_PRIMITIVE,1); // ...❶
arr = pointSprite.getVertexArray();          // ...❷
arr[0]=80; arr[1]=0; arr[2]=0;               // 頂点(80,0,0)
arr = pointSprite.getPointSpriteArray();     // ...❹
arr[0]=48; arr[1]=48; arr[2]=0; // 幅x高さx回転角(48x48x0)
arr[3]= 0; arr[4]= 0;           // 左上テクスチャ位置(0,0)
arr[5]=48; arr[6]=48;           // 右下テクスチャ位置(48,48)
arr[7]=Primitive.POINT_SPRITE_FLAG_PIXEL_SIZE| // フラグ
    Primitive.POINT_SPRITE_FLAG_PERSPECTIVE;
```

```
        // テクスチャの指定 ...❺
        try {
            Texture texture = (Texture)Object3D.createInstance(
                Connector.openInputStream("resource:///jyagi.bmp"));
            pointSprite.setTexture(texture);
        } catch (Exception e) {
        }

        // 視点座標の指定
        trans.lookAt(
            new Vector3D(0,0,100), // 視点
            new Vector3D(0,0,0),   // 参照点
            new Vector3D(0,1,0));  // UPベクトル

        while (true) {
            // キー状態
            keyState = getKeypadState();
            if (((1<<Display.KEY_UP)&keyState) != 0) {
                trans.rotate(1,0,0,16);
            } else if (((1<<Display.KEY_DOWN)&keyState) != 0) {
                trans.rotate(1,0,0,-16);
            } else if (((1<<Display.KEY_LEFT)&keyState) != 0) {
                trans.rotate(0,1,0,16);
            } else if (((1<<Display.KEY_RIGHT)&keyState) != 0) {
                trans.rotate(0,1,0,-16);
            }

            // 回転角度の指定
            g3.setTransform(trans);

            // プリミティブの表示 ...❻
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g3.renderObject3D(point,origin);
            g3.renderObject3D(line,origin);
            g3.renderObject3D(triangle,origin);
            g3.renderObject3D(quad,origin);
            g3.renderObject3D(pointSprite,origin);
            g3.flushBuffer();
            g.unlock(true);

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }
```

Next page. ↓

```
    // 描画
    public void paint(Graphics g) {}
}
```

DoJa PrimitiveCanvas.java

ⓐ

```
import com.nttdocomo.ui.graphics3d.Graphics3D;
import com.nttdocomo.ui.graphics3d.Object3D;
import com.nttdocomo.ui.graphics3d.Primitive;
import com.nttdocomo.ui.graphics3d.Texture;
import com.nttdocomo.ui.util3d.Transform;
import com.nttdocomo.ui.util3d.Vector3D;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import javax.microedition.io.Connector;
```

PrimitiveCanvas ...❶：プリミティブの生成

プリミティブ情報は**Primitiveクラス**で保持します。引数に*type*（プリミティブの型）と*param*（プリミティブの種類）を指定してオブジェクトを生成した後、get()メソッドによって取得する配列を操作することでプリミティブ情報を編集します。Primitiveクラスのコンストラクタは次の通りです。

［Primitiveクラス］

Primitive(*type, param, num*)

［解説］ Primitiveクラスのコンストラクタ
［引数］ *type* プリミティブの型：int型
param プリミティブの種類：int型
num プリミティブの数（最大255）：int型

引数*type*（プリミティブの型）には次の定数を指定します。

Primitive.PRIMITIVE_POINTS	点
Primitive.PRIMITIVE_LINES	線
Primitive.PRIMITIVE_TRIANGLES	三角形
Primitive.PRIMITIVE_QUADS	四角形
Primitive.PRIMITIVE_POINT_SPRITES	ポイントスプライト

引数の*param*（プリミティブの種類）には次の定数を論理和で指定します。

色	Primitive.COLOR_NONE	色情報を持たない
	Primitive.COLOR_PER_PRIMITIVE	プリミティブ単位で同じ色情報を持つ
	Primitive.COLOR_PER_FACE	面単位で色情報を持つ
法線	Primitive.NORMAL_NONE	法線情報を持たない
	Primitive.NORMAL_PER_FACE	面単位で法線情報を持つ
	Primitive.NORMAL_PER_VERTEX	頂点単位で法線情報を持つ
テクスチャ	Primitive.TEXTURE_COORD_NONE	テクスチャ座標情報を持たない
	Primitive.TEXTURE_COORD_PER_VERTEX	頂点単位でテクスチャ座標情報を持つ
ポイントスプライト	Primitive.POINT_SPRITE_PER_PRIMITIVE	プリミティブ単位で同じポイントスプライト情報を持つ
	Primitive.POINT_SPRITE_PER_VERTEX	頂点単位でポイントスプライト情報を持つ

PrimitiveCanvas ...❷：頂点情報の指定

頂点情報はgetVertexArray()メソッドで取得します。全てのプリミティブで使用します。

［Primitiveクラス］

int[] getVertexArray()

［解説］　頂点情報の取得

［戻り値］　頂点情報

戻り値の頂点情報は、プリミティブ1個につき、点は頂点1個、線は頂点2個、三角形は頂点3個、四角形は頂点4個、ポイントスプライトは頂点1個を格納します。頂点1個につき（x, y, z）の順で3個の整数値を格納します。配列の長さは、点はプリミティブ数x3、線はプリミティブ数x6、三角形はプリミティブ数x9、四角形はプリミティブ数x12、ポイントスプライトはポイントスプライト数x3となります。

chapter

1
2
3
4
5
6
7
8
a

PrimitiveCanvas ...❸：色情報の指定

色情報はgetColorArray()メソッドで取得します。点と線と三角形と四角形で使用します。

［Primitiveクラス］

int[] getColorArray()

［解説］　色情報の取得
［戻り値］　色情報

戻り値は、コンストラクタの引数で指定した*param*（プリミティブの種類）によって変ります。

Primitive.COLOR_NONE	null
Primitive.COLOR_PER_PRIMITIVE	プリミティブの個数に関わらず、共通に使われる色情報1個を格納
	色情報1個につきRGB値を表す1個の整数値を格納
	配列の長さは常に1
Primitive.COLOR_PER_FACE	プリミティブ1個につき色情報1個を格納
	情報1個につきRGB値を表す1個の整数値を格納
	配列の長さはプリミティブ数x1

Column»　法線情報の指定

法線情報はgetNormalArray()メソッドで取得します。三角形と四角形で使用します。

［Primitiveクラス］

int[] getNormalArray()

［解説］　法線情報の取得
［戻り値］　法線情報

戻り値は、コンストラクタの引数で指定した*param*（プリミティブの種類）によって変ります。

Primitive.NORMAL_NONE	null
Primitive.NORMAL_PER_FACE	プリミティブ1個につき法線情報1個を格納
	法線情報1個につき（x, y, z）の順で3個の整数値を格納
	配列の長さは「プリミティブ数x3」
Primitive.NORMAL_PER_VERTEX	プリミティブ1個につき三角形は法線情報3個、四角形は法線情報4個を格納
	法線情報1個につき（x, y, z）の順で3個の整数値を格納
	配列の長さは三角形は「プリミティブ数x9」、四角形は「プリミティブ数x12」

ちなみに今回のサンプルでは法線を利用していません。

テクスチャ情報の指定

テクスチャ情報はgetTextureCoordArray()メソッドで取得します。三角形と四角形で使用します。

［Primitiveクラス］

int[] getTextureCoordArray()

［解説］　テクスチャ情報の取得
［戻り値］　テクスチャ情報

戻り値は、コンストラクタの引数で指定した*param*（プリミティブの種類）によって変ります。

Primitive.TEXTURE_COORD_NONE	null
Primitive.TEXTURE_COORD_PER_VERTEX	プリミティブ1個につき三角形はテクスチャ座標情報3個
	四角形はテクスチャ座標情報4個を格納、テクスチャ座標情報1個につき（u,v）の順で2個の整数値を格納
	配列の長さは三角形は「プリミティブ数x6」、四角形は「プリミティブ数x8」

ちなみに今回のサンプルではテクスチャ情報は利用していません。

PrimitiveCanvas …❹：ポイントスプライト情報の指定

ポイントスプライト情報はgetPointSpriteArray()メソッドで取得します。ポイントスプライトで使用します。

［Primitiveクラス］

int[] getPointSpriteArray()

［解説］　ポイントスプライト情報の取得
［戻り値］　ポイントスプライト情報

戻り値は、コンストラクタの引数で指定した*param*（プリミティブの種類）によって変ります。

Primitive.POINT_SPRITE_PER_PRIMITIVE	プリミティブの個数に関わらず、共通に使われるプリミティブ情報1個を格納
	ポイントスプライト情報1個につき、ポイントスプライトの幅・高さ・回転角・ポイントスプライトの左上に対応するテクスチャ位置情報（u0, v0）・右下に対応するテクスチャ位置情報（u1, v1)表示フラグの8個の整数値を格納
	配列の長さは常に8
Primitive.POINT_SPRITE_PER_VERTEX	プリミティブ1個につきポイントスプライト情報1個を格納
	ポイントスプライト情報1個につき、ポイントスプライトの幅・高さ・回転角・ポイントスプライトの左上に対応するテクスチャ位置情報（u0, v0）・右下に対応するテクスチャ位置情報（u1, v1）・表示フラグの8個の整数値を格納
	配列の長さは「プリミティブ数x8」

表示フラグにはGraphics3Dインタフェースが持つ次の定数から必要なものを論理和で指定します。

Primitive.POINT_SPRITE_FLAG_LOCAL_SIZE	モデル座標系でのサイズを指定
Primitive.POINT_SPRITE_FLAG_PIXEL_SIZE	スクリーン座標系でのサイズを指定
Primitive.POINT_SPRITE_FLAG_PERSPECTIVE	パースを有効にする
Primitive.POINT_SPRITE_FLAG_NO_PERSPECTIVE	パースを無効にする

PrimitiveCanvas ...❺：テクスチャの指定

プリミティブにテクスチャを指定するには、PrimitiveクラスのsetTexture()メソッドを使います。

［Primitiveクラス］

void setTexture(*texture*)

［解説］ テクスチャの指定

［引数］ *texture* テクスチャ：Texture型

テクスチャは**Object3Dクラス**のcreateInstance()メソッドで読み込み、Textureクラスにキャストして利用します。リソースを入力ストリーム経由で読み込むには、Connector.openInputStream()メソッドを使います。

PrimitiveCanvas ...❻：プリミティブの表示

プリミティブを表示するには、モデルと同様にGraphics3Dインタフェースのrender0bject3D()メソッドを使います。render0bject3D()メソッドで行うのはレンダリング演算のみで、実際に描画を行うにはGraphics3Dクラスのflush Buffer()メソッドを呼ぶ必要があります。

chapter
1
2
3
4
5
6
7
8
a

6-3 平行投影と透視投影

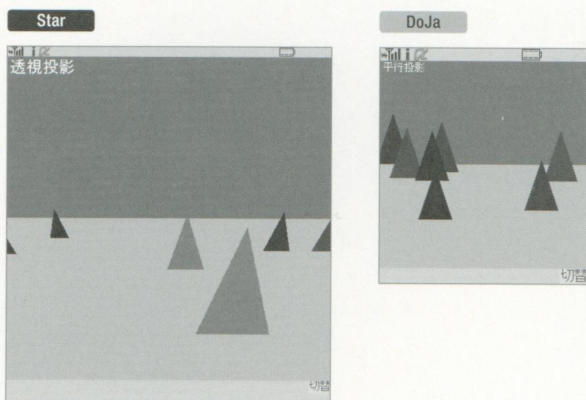


Star
http://book.mycom.co.jp/support/pc/star/63s.html



DoJa
http://book.mycom.co.jp/support/pc/star/d/63d.html

平行投影と**透視投影**を切り替えて表示するプログラムを作ります。



ソフトキー 2	投影方法の変更

このプログラムは、次の2つのクラスで構成されています。

- **ProjectionExクラス**（**ProjectionEx.java**）
- **ProjectionCanvasクラス**（**ProjectionCanvas.java**）

プロジェクト名「**ProjectionEx**」でプロジェクトを作成してください。

▶ProjectionExクラス

ProjectionExクラスは、プログラムの本体となるクラスです。

Star ProjectionEx.java

```
import com.docomostar.StarApplication; // ⓐ
```

```
import com.docomostar.ui.Display;       //

// 平行投影と透視投影（本体）
public class ProjectionEx extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {           // ⓒ
        ProjectionCanvas c = new ProjectionCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa ProjectionEx.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class ProjectionEx extends IApplication {
```

ⓒ

```
    public void start() {
```

▶ProjectionCanvasクラス

ProjectionCanvasクラスはキャンバスとなるクラスです。

Star ProjectionCanvas.java

```
import com.docomostar.ui.graphics3d.Graphics3D; // ⓐ
import com.docomostar.ui.graphics3d.Primitive;  //
import com.docomostar.ui.util3d.Transform;      //
import com.docomostar.ui.util3d.Vector3D;       //
import com.docomostar.ui.Canvas;                //
import com.docomostar.ui.Display;               //
import com.docomostar.ui.Graphics;              //
import javax.microedition.io.Connector;
import java.util.Random;

// 平行投影と透視投影（キャンバス）
```

Next page. ⬇

```
public class ProjectionCanvas extends Canvas
    implements Runnable {
    // 頂点
    private final static int[] TREE_V = {  // 木 - 三角形
        -40,30,0, -60,-30,0, -20,-30,0};

    // システム
    private Graphics   g  = getGraphics(); // グラフィックス
    private Graphics3D g3 = (Graphics3D)g; // 3D グラフィックス
    private int        keyEvent = -999;    // キーイベント

    // プリミティブ
    private Primitive tree;              // 木
    private int[] treeX = new int[32];   // 木の X 座標
    private int[] treeZ = new int[32];   // 木の Z 座標
    private int[] treeC = new int[32];   // 木の色

    // 座標
    private Transform trans = new Transform();  // 視点座標
    private Transform origin = new Transform(); // 原点

    // 処理
    public void run() {
        int i,j,mode=0;
        Random rand = new Random();

        // 木
        tree = new Primitive(
            Primitive.PRIMITIVE_TRIANGLES,
            Primitive.COLOR_PER_PRIMITIVE|
            Primitive.NORMAL_NONE,1);
        for (i=0;i<TREE_V.length;i+=3) {
            tree.getVertexArray()[i]   = TREE_V[i  ];
            tree.getVertexArray()[i+1] = TREE_V[i+1];
            tree.getVertexArray()[i+2] = TREE_V[i+2];
        }
        for (i=0;i<32;i++) treeZ[i] = -9999;          // ⓑ

        // 視点座標の指定
        trans.lookAt(
            new Vector3D(0,40,220), // 視点
            new Vector3D(0,5,0),    // 参照点
            new Vector3D(0,1,0));   // UP ベクトル
        g3.setTransform(trans);

        // ソフトラベル
        setSoftLabel(Canvas.SOFT_KEY_2," 切替 ");

        while (true) {
```

```
// キーイベント
if (keyEvent==Display.KEY_SOFT2) {
    mode = (mode==0)?1:0;
    // 平行投影の指定 …❶
    if (mode==0) {
        g3.setParallelView(480,480);         // ⓒ
    }
    // 透視投影の指定 …❷
    else {
        g3.setPerspectiveView(1,200,120);
    }
}
keyEvent = -999;

// 三角形の移動と出現
for (j=0;j<32;j++) {
    // 移動
    if (treeZ[j] != -9999) {                 // ⓓ
        treeZ[j] += 10;                      //
        if (treeZ[j]>220) treeZ[j] = -9999;  //
    }
    // 出現
    else if ((rand.nextInt()>>>1)%20==0) {
        treeX[j] = (rand.nextInt()>>>1)%600-300;
        treeZ[j] = -260;
        treeC[j] = ((rand.nextInt()>>>1)%155)<<8;
    }
}

// 背景の表示
g.lock();
g.setColor(g.getColorOfRGB(100,100,255));
g.fillRect(0,0,480,240);                     // ⓔ
g.setColor(g.getColorOfRGB(96,224,96));
g.fillRect(0,240,480,getHeight()-240);       // ⓕ

// 三角形の表示
for (j=0;j<32;j++) {
    tree.getColorArray()[0] = treeC[j];
    origin.setIdentity();
    origin.translate(treeX[j],0,treeZ[j]);
    g3.renderObject3D(tree,origin);
}
g3.flushBuffer();

// 投影方法の表示
g.setColor(g.getColorOfName(g.WHITE));
if (mode==0) g.drawString("平行投影",4,24); // ⓖ
else         g.drawString("透視投影",4,24); //
```

Next page. ⬇

```
                g.unlock(true);

                // スリープ
                try {
                    Thread.sleep(100);
                } catch (Exception e) {
                }
            }
        }

        // キーイベント
        public void processEvent(int type,int param) {
            if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
        }

        // 描画
        public void paint(Graphics g) {}
    }
```

DoJa ProjectionCanvas.java

ⓐ

```
import com.docomostar.ui.graphics3d.Graphics3D;
import com.docomostar.ui.graphics3d.Primitive;
import com.docomostar.ui.util3d.Transform;
import com.docomostar.ui.util3d.Vector3D;
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Display;
import com.docomostar.ui.Graphics;
```

ⓑ

```
        for (i=0;i<32;i++) treeZ[i] = -9999;
```

ⓒ

```
                g3.setParallelView(480,480);
```

ⓓ

```
                if (treeZ[j] != -9999) {
                    treeZ[j] += 10;
                    if (treeZ[j]>220) treeZ[j] = -9999;
                }
```

ⓔ

```
            g.fillRect(0,0,480,240);
```

ⓕ

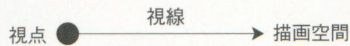
```
            g.fillRect(0,240,480,getHeight()-240);
```

ⓖ

```
            if (mode==0) g.drawString("平行投影",4,24);
            else         g.drawString("透視投影",4,24);
```

ProjectionCanvas ...❶：平行投影の指定

平行投影は、モデルと投影面を結ぶ線がそれぞれ平行となる投影法です。モデルのサイズは視点の距離に左右されません。**Graphics3Dインタフェース**のsetParallelView()メソッドを使うことで平行投影になります。



［Graphics3Dインタフェース］

void setParallelView(*width, height*)

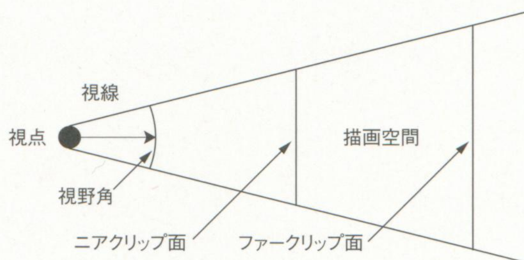
［解説］ 平行投影の指定

［引数］ *width* 投影面の幅：int型

height 投影面の高さ：int型

ProjectionCanvas ...❷：透視投影の指定

透視投影は、視点から遠くのものは小さく、近くのものは大きく表示することにより、遠近感をだす投影法です。モデルのサイズは視点からの距離に左右されます。視点から近すぎる、遠すぎる、視線とは反対方向にあるモデルは描画されません。描画される空間の視点から最も近い面を**ニアクリップ面**、最も遠い面を**ファークリップ面**といいます。透視投影を使うには、Graphics3Dインタフェースのsetl PerspectiveView()メソッドを使います。



［Graphics3Dインタフェース］

void setPerspectiveView(*zNear, zFar, angle*)

［解説］ 透視投影の指定

［引数］ *zNear* カメラからニアクリップ面までの距離：int型
zFar カメラからファークリップ面までの距離：int型
angle 視野角（0〜180度）：int型

［Graphics3Dインタフェース］

void setPerspectiveView(*zNear, zFar, width, height*)

［解説］ 透視投影の指定

［引数］ *zNear* カメラからニアクリップ面までの距離：int型
zFar カメラからファークリップ面までの距離：int型
width ニアクリップ面における投影面の幅：int型
height ニアクリップ面における投影面の高さ：int型

Star/DoJaの独自機能

chapter

1
2
3
4
5
6
7
8
a

本章では、iウィジェットやJava/Flash連携などのStarプロファイルの新機能と、DoJaプロファイルでのみ作成できる待ち受けアプリの作り方について解説します。

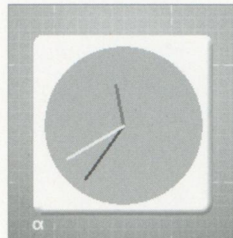
7-1 iウィジェット



Star
http://book.mycom.co.jp/support/pc/star/71s.html
※ **iウィジェット対応機種のみ**
F-01A、03A、N-01A、02A、04A、SH-01A、03A、04A、P-01A、02Aに対応（2009年2月現在）

時刻を表示する**iウィジェット**を作ります。この機能はStarプロファイルの基本APIです。DoJaプロファイルでは利用できません。ユーザが端末設定でウィジェットビューに貼り付けることにより**ウィジェット**として起動できます。





ソフトキー１	デジタル表示とアナログ表示の切り替え

iウィジェットは最大8個まで同時起動できるミニアプリです。端末のiウィジェットボタンを押すことでiウィジェットが並んでいる画面**ウィジェットビュー**が表示されます。ユーザが好きな時に手軽に見ることができるという点がメリットとなります。

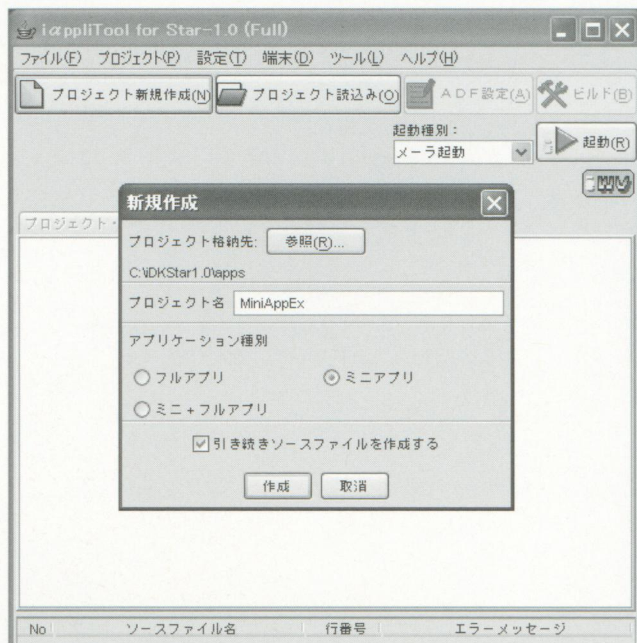
通常iアプリとウィジェットを区別するために、通常iアプリを**フルアプリ**、ウィジェットを**ウィジェットアプリ／ミニアプリ**とも呼びます。

このプログラムは、次の2つのクラスで構成されています。

- **MiniAppExクラス（MiniAppEx.java）**
- **MiniAppCanvasクラス（MiniAppCanvas.java）**

プロジェクト名「**MiniAppEx**」でプロジェクトを作成してください。

［プロジェクト新規作成］ボタンをクリックした後、プロジェクト名に「MiniAppEx」と入力し、アプリケーション種別として「ミニアプリ」を選択してウィジェット用のプロジェクトを作成してください。



Column»　ミニアプリ+フルアプリ

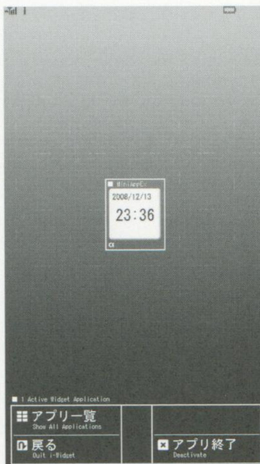
新規作成ダイアログで「**ミニ+フルアプリ**」でプロジェクトを生成すると、ミニアプリとフルアプリの両方を含んだJARファイルを作成することができます。ウィジェットの機能制限により通常ミニアプリからフルアプリの起動はできませんが、同一パッケージのフルアプリであれば起動することができます。

▶ウィジェットビューの状態

ウィジェットビューには**一覧表示状態**、**個別表示状態**、**ランチャー表示状態**の3つの状態が存在します。

・一覧表示状態

起動中（Active）のウィジェットが複数表示されている画面です。方向キーでウィジェットの選択し、決定キーで個別表示状態に遷移します。この状態の時は、ウィジェットは縮小表示され、キーイベントも通知されません。



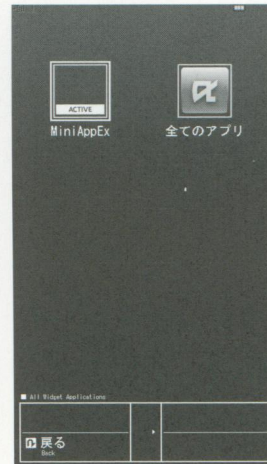
・個別表示状態

特定の1つのウィジェットを選択表示する画面です。キーイベントはウィジェットに通知されます。



・ランチャー表示状態

ウィジェットビューに貼り付けるウィジェットを選択する画面です。ウィジェットのアイコン一覧が表示されています。



▶ウィジェットの状態

ウィジェットは**Focused状態**、**Unfocused状態**、**Selected状態**、**Unselected状態**の4つの状態を遷移します。

Focused状態	一覧表示状態で選択されているウィジェットの状態
Unfocused状態	一覧表示状態で選択されていないウィジェットの状態
Selected状態	個別表示状態で選択されているウィジェットの状態
Unselected状態	個別表示状態で選択されていないウィジェットの状態

StarApplicationクラスのaddEventListener()メソッドでイベントリスナーを登録することにより、ウィジェットの状態の遷移を知ることができます。

STAR_FACE_STATECHANGE_MINI_FOCUSED	Focused状態への遷移
STAR_FACE_STATECHANGE_MINI_UNFOCUSED	Unfocused状態への遷移
STAR_FACE_STATECHANGE_MINI_SELECTED	Selected状態への遷移

▶ウィジェットの機能制限

ウィジェットはフルアプリの機能を全て使えるわけではありません。主な機能制限は次の通りです。

- JARファイルのサイズは50KB
- スクラッチパッドのサイズは200KB、アクセスできるのは0番("scratchpad:///0")のみ
- 画面サイズは440x80、320x240、240x320、160x160の4種類のみ
- 高レベルUIは不可（Panel/Component）
- ダイアログは不可（Dialog）
- プレゼンターは不可（AudioPresenter/VisualPresenter）
- iモーション再生は不可（NativeMoviePlayer）
- 3Dグラフィックス／3Dサウンドは不可
- イメージエンコーダは不可
- iアプリ連携起動は不可
- 電話帳／ブックマーク／スケジューラ／カメラ呼び出しは不可
- イメージデータ／映像データ／データBOXの管理機能呼び出しは不可
- メモリ管理機能は不可（MemoryManager）

▶MiniAppExクラス

MiniAppExクラスは、プログラムの本体となるクラスです。

Star MiniAppEx.java

```
import com.docomostar.StarApplication;
import com.docomostar.ui.Display;

// iウィジェット(本体)
public class MiniAppEx extends StarApplication {

    // アプリの開始
```

```
    public void started(int launchType) {
        MiniAppCanvas c = new MiniAppCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

▶MiniAppCanvasクラス

MiniAppCanvasクラスはキャンバスとなるクラスです。

Star MiniAppCanvas.java

```
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Display;
import com.docomostar.ui.Font;
import com.docomostar.ui.Graphics;
import java.util.Calendar;

// iウィジェット(キャンバス)
public class MiniAppCanvas extends Canvas
    implements Runnable {
    // 回転定数 ...❶
    private final static int DX[] = {
        0,426,848,1261,1666,2046,2405,2737,3043,3312,
        3545,3739,3895,4006,4073,4096,4074,4007,3897,3742,
        3548,3316,3048,2741,2410,2052,1671,1267,854,433,
        0,-426,-848,-1261,-1666,-2046,-2405,-2737,-3043,-3312,
        -3545,-3739,-3895,-4006,-4073,-4096,-4074,-4007,-3897,-3742,
        -3548,-3316,-3048,-2741,-2410,-2052,-1671,-1267,-854,-433};
    private final static int DY[] = {
        -4096,-4074,-4007,-3897,-3742,-3548,-3316,-3048,-2741,-2410,
        -2052,-1671,-1267,-854,-433,0,426,848,1261,1666,
        2046,2405,2737,3043,3312,3545,3739,3895,4006,4073,
        4096,4074,4007,3897,3742,3548,3316,3048,2741,2410,
        2052,1671,1267,854,433,0,-426,-848,-1261,-1666,
        -2046,-2405,-2737,-3043,-3312,-3545,-3739,-3895,-4006,-4073};

    // 変数
    private int keyEvent = -999; // キーイベント
    private Graphics g;          // グラフィックス

    // 処理
    public void run() {
        g = getGraphics();
```

Next page. ⬇

chapter

1

2

3

4

5

6

7

8

a

```
// 変数
Calendar cal; // カレンダー
int year;     // 年
int month;    // 月
int day;      // 日
int hour;     // 時
int minute;   // 分
int second;   // 秒
int mode = 0; // モード(0:デジタル,1:アナログ)

// ソフトキー
setSoftLabel(SOFT_KEY_1,"モード");

while (true) {
    // 時刻の取得
    cal    = Calendar.getInstance();
    year   = cal.get(Calendar.YEAR);
    month  = cal.get(Calendar.MONTH)+1;
    day    = cal.get(Calendar.DAY_OF_MONTH);
    hour   = cal.get(Calendar.HOUR_OF_DAY);
    minute = cal.get(Calendar.MINUTE);
    second = cal.get(Calendar.SECOND);

    // 描画
    g.lock();
    g.setColor(g.getColorOfName(g.WHITE));
    g.fillRect(0,0,getWidth(),getHeight());

    // デジタル
    if (mode==0) {
        g.setColor(g.getColorOfName(g.BLACK));
        g.setFont(Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,24));
        g.drawString(year       +"/"+
            formatNum(month," ")+"/"+
            formatNum(day," "),0,24);
        g.setFont(Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,48));
        g.drawString(
            formatNum(hour,  "0")+":"+
            formatNum(minute,"0"),10,90);
    }

    // アナログ
    else {
        g.setColor(g.getColorOfName(g.SILVER));
        g.fillArc(0,0,140,140,0,360);

        g.setColor(g.getColorOfName(g.RED));
```

```
                drawLine(70,70,
                    70+DX[hour%12*5+minute/12]/110,
                    70+DY[hour%12*5+minute/12]/110);
                g.setColor(g.getColorOfName(g.BLUE));
                drawLine(70,70,70+DX[minute]/70,70+DY[minute]/70);
                g.setColor(g.getColorOfName(g.YELLOW));
                drawLine(70,70,70+DX[second]/70,70+DY[second]/70);
            }

            // アンロック
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_SOFT1) {
                mode = (mode+1)%2;
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // ラインの描画
    private void drawLine(int x0,int y0,int x1,int y1) {
        g.drawLine(x0,y0,x1,y1);
        g.drawLine(x0+1,y0,x1+1,y1);
        g.drawLine(x0-1,y0,x1-1,y1);
        g.drawLine(x0,y0+1,x1,y1+1);
        g.drawLine(x0,y0-1,x1,y1-1);
    }

    // 数字書式を整える
    private String formatNum(int num,String ume) {
        if (num<10) {
            return ume+num;
        } else {
            return ""+num;
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }
```

Next page. ↓

```
    // 描画
    public void paint(Graphics g) {}
}
```

MiniAppCanvas ...❶：回転定数

回転定数はアナログ時計の針の描画に利用する位置情報です。配列の長さは60で、12時の方向から時計回りに6度ずつ移動した時のXY座標を、int DX[]、int DY[]にそれぞれ保持しています。

▶ADFの設定

フルアプリでは「画面サイズ ＝ 描画領域」でしたが、ウィジェットではそうではありません。ウィジェットでは背景となるイメージと描画領域の位置を指定することで、角の丸いウィジェットを作れるようにしています。

今回は160x160の角の丸い背景イメージを用意し、その中のXY座標（10, 10)、幅高さ140x140の領域を描画領域として指定しています。背景イメージは「res」フォルダの直下にリソースとして配置してください。

・背景イメージ：**palette.gif**



ウィジェット全体のサイズのことを**パレットサイズ**、その中の描画領域のサイズのことを**パレット描画領域サイズ**、背景イメージのことを**パレットフェイス**と呼びます。

これらの情報はADFの「Pallet」で設定します。

Pallet	160x160:140x140:10:10:palette.gif

Palletパラメータの書式は次の通りです。

```
<PalletSize>:<PalletDrawArea>:<PositionX>:<PositionY>:<PalletFace>
```

・<PaletteSize>

パレットサイズを次の書式で指定します。指定可能なサイズは「320x240」「240x320」「160x160」「440x80」の4種類です。

```
<width>x<height>
```

・<PalletDrawArea>

パレット上の描画領域のサイズを次の書式で指定します。指定可能なサイズはPaletteSizeより小さなサイズです。

```
<width>x<height>
```

・<PositionX>,<PositionY>

パレット上の描画領域の位置を指定します。

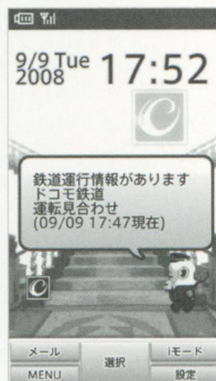
・<PaleteFace>

パレットフェイスとなるイメージのファイル名を指定します。

Column» iコンシェル

iコンシェルはユーザに関する情報を収集・分析して、ユーザが必要であろうと推測して情報を配信してくれるサービスです（利用料は月額210円です）。

たとえばユーザの住所を元に路線や天気の情報を提供したりします。情報は待ち受け画面のキャラクタが知らせてくれます。iコンシェルの「コンシェル」の用語は、ホテルなどで宿泊客のさまざまな要望に対応する係「コンシェルジュ」からきています。



・iコンシェル
http://answer.nttdocomo.co.jp/concier/

2009年1月現在の対応機種は次の13機種です。

・F-01A/ F-03A
・N-01A/ N-02A/ N-04A
・P-01A/ P-02A/ P-03A/ P-04A/ P-05A
・SH-01A/ SH-03A/ SH-04A

iコンシェルはiアプリでなくマチキャラをベースに作成されているので、オリジナルのiコンシェルのキャラクターを作るにはマチキャラの作成ツールを使います。詳しくは以下のサイトを参照してください。

・作ろうiモードコンテンツ　iコンシェル
http://www.nttdocomo.co.jp/service/imode/make/content/iconcier/

Column »　iアプリオンラインとiアプリコール

iアプリオンラインは、iアプリで複数人とリアルタイムに通信を行えるサービスです。iアプリは基本的にダウンロード元サーバとのHTTP/HTTPS通信しかサポートしていませんが、iアプリDXの機能によりTCP/UDPによるソケット通信も許可され、通信先も制限なく端末同士のPeer to Peer型通信も可能となっています。

・iアプリオンライン
http://answer.nttdocomo.co.jp/concier/

さらには**iアプリコール**と呼ばれるiアプリから別のユーザを招集する機能も追加されています。電話帳のデータまたはコンテンツプロバイダのサーバ上にあるリストから召集する相手を選択するというものです。
2009年1月現在の対応機種は次の10機種です。

・N-01A/ N-02A/ N-04A
・P-01A/ P-02A
・F-01A/ F-03A
・SH-01A/ SH-03A/ SH-04A

7-2 Java-Flash連携機能

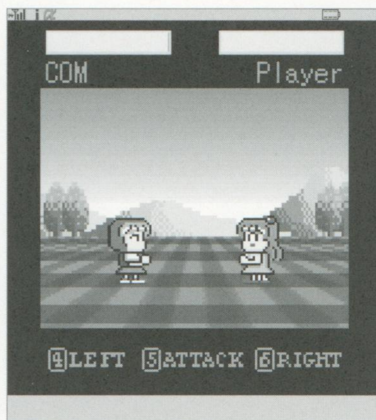


Star

http://book.mycom.co.jp/support/pc/star/72s.html

※ StarプロファイルのオプションAPI
F-01A、03A、N-01A、02A、04A、SH-01A、03A、04A、P-01A、02Aに対応（2009年2月現在）

iアプリ上で**Flashコンテンツ**を再生するプログラムを作ります。この機能はStarプロファイルのオプションAPIです。



FlashとはAdobe Systemsが開発したWebコンテンツの作成ソフトです。作成したコンテンツのこともFlashと呼びますが、同じではわかりにくいので、本書ではFlashコンテンツと呼ぶことにします。NTTドコモAシリーズで再生可能なFlashのバージョンは**Flash Lite 3**です。詳しくは以下のサイトを参照してください。

- **作ろうiモードコンテンツ - Flash**
  http://www.nttdocomo.co.jp/service/imode/make/content/flash/

このプログラムは、次の1つのクラスで構成されています。

- **FlashExクラス（FlashEx.java）**

プロジェクト名「**FlashEx**」でプロジェクトを作成してください。

▶Flashコンテンツの準備

今回使用するFlashコンテンツファイルは次の1点です。プロジェクトの「res」フォルダに置いてください。

Flashコンテンツの作成方法については本書の範囲外なので解説は省略します。

・Flashコンテンツ：**kakuto.swf**（Flash Lite 3用コンテンツ）



▶FlashExクラス

FlashExクラスは、プログラムの本体となるクラスです。

Star FlashEx.java

```
import com.docomostar.StarApplication;
import com.docomostar.ui.Display;
import com.docomostar.ui.flashplayer.FlashPlayerPane;
import javax.microedition.io.Connector;
import java.io.InputStream;

// Java-Flash 連携機能 ( 本体 )
public class FlashEx extends StarApplication {

    // アプリの開始
    public void started(int launchType) {
        try {
```

```
            // Flash コンテンツの読み込み ...❶
            byte[] w = new byte[102400];
            InputStream in = Connector.openInputStream("resource:///kakuto.swf");
            int size = in.read(w);
            in.close();

            // Flash コンテンツの再生 ...❷
            FlashPlayerPane pane = new FlashPlayerPane();
            pane.set(w);
            Display.setCurrent(pane);
            pane.play();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```


FlashEx ...❶：Flashコンテンツの読み込み

ConnectorクラスのopenInputStream()メソッドでFlashコンテンツを読み込みます。

［Connectorクラス］

static InputStream openInputStream(*location*)

［解説］ リソースの読み込み

［引数］ *location* 読み込み先：String型

［戻り値］ 入力ストリーム

FlashEx ...❷：Flashコンテンツの再生

Flashコンテンツを再生するには、**FlashPlayerPaneクラス**を使います。FlashPlayerPaneオブジェクトを生成後、set()メソッドでFlashコンテンツのバイトデータを指定します。

［FlashPlayerPaneクラス］

void set(*data*)

［解説］ Flashコンテンツを指定

［引数］ *data* Flashコンテンツのバイトデータ：byte[]型

背景色を指定するset()メソッドもあります（次ページ）。

［FlashPlayerPaneクラス］

void set(*data, rgb*)

［解説］ Flashコンテンツを指定

［引数］ *data* Flashコンテンツのバイトデータ：byte[]型

rgb 背景色：int型

そしてDisplayクラスのsetCurrent()メソッドで画面に設定後、FlashPlayerPaneクラスのplay()メソッドで再生します。

［FlashPlayerPaneクラス］

void play()

［解説］ Flashコンテンツの再生

7-3 タッチイベントの処理

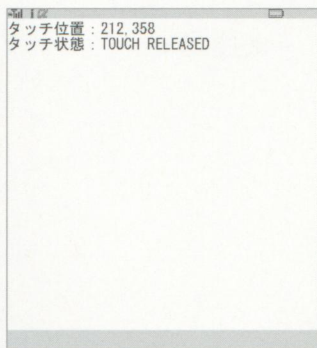


Star

http://book.mycom.co.jp/supoort/pc/star/73s.html

※ StarプロファイルのオプションAPI
N-01A、SH-03A、04Aに対応（2009年2月現在）

タッチイベントの処理を行うプログラムを作ります。この機能はStarプロファイルのオプションAPIです。



このプログラムは、次の2つのクラスで構成されています。

- **TouchExクラス（TouchEx.java）**
- **TouchCanvasクラス（TouchCanvas.java）**

プロジェクト名「**TouchEx**」でプロジェクトを作成してください。

▶TouchExクラス

TouchExクラスは、プログラムの本体となるクラスです。

Star TouchEx.java

```
import com.docomostar.StarApplication;
import com.docomostar.ui.Display;

// タッチイベントの処理(本体)
public class TouchEx extends StarApplication {
```

```
    // アプリの開始
    public void started(int launchType) {
        TouchCanvas c = new TouchCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

▶TouchCanvasクラス

TouchCanvasクラスはキャンバスとなるクラスです。

Star TouchCanvas.java

```
import com.docomostar.opt.ui.TouchDevice;
import com.docomostar.ui.Canvas;
import com.docomostar.ui.Display;
import com.docomostar.ui.Graphics;

// タッチイベントの処理（キャンバス）
public class TouchCanvas extends Canvas
    implements Runnable {
    private String info = ""; // 情報

    // 処理
    public void run() {
        // グラフィック
        Graphics g = getGraphics();

        // タッチイベントの有効化 …❶
        TouchDevice.setEnabled(true);

        while (true) {
            // タッチ位置の取得 …❷
            int touchX = TouchDevice.getX();
            int touchY = TouchDevice.getY();

            // 画面の描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());
            g.setColor(g.getColorOfName(g.BLACK));
            g.drawString("タッチ位置："+touchX+","+touchY,0,24*1);
            g.drawString("タッチ状態："+info,0,24*2);
            g.unlock(true);

            // スリープ
            try {
```

```
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // タッチ状態の取得 ...❸
    public void processEvent(int type,int param) {
        if (type==Display.TOUCH_PRESSED_EVENT) {
            info = "TOUCH PRESSED";
        } else if (type==Display.TOUCH_MOVEDSTART_EVENT) {
            info = "TOUCH MOVEDSTART";
        } else if (type==Display.TOUCH_MOVEDEND_EVENT) {
            info = "TOUCH MOVEDEND";
        } else if (type==Display.TOUCH_RELEASED_EVENT) {
            info = "TOUCH RELEASED";
        }
    }

    // 描画
    public void paint(Graphics g) {}
}
```

TouchCanvas ...❶：タッチイベントの有効化

タッチイベントを処理するには、まずタッチイベントを有効化する必要があります。タッチイベントを有効化するには**TouchDeviceクラス**のsetEnabled()メソッドを使います。ただし、端末設定でタッチパネルデバイスが使用不可能に設定されている場合は有効になりません。

［TouchDeviceクラス］

static void setEnabled(*flag*)

［解説］　タッチパネルの有効無効を指定

［引数］　*flag*　タッチパネルの有効無効:boolean型

TouchCanvas ...❷：タッチ位置の取得

最終タッチ位置は**TouchDevice**クラスのgetX()メソッドとgetY()メソッドで取得します。

［TouchDeviceクラス］

static int getX()

［解説］　最終タッチのX座標の取得
［戻り値］　最終タッチのX座標

［TouchDeviceクラス］

static int getY()

［解説］　最終タッチのY座標の取得
［戻り値］　最終タッチのY座標

TouchCanvas ...❸：タッチ状態の取得

タッチイベントを有効化すると、CanvasクラスのprocessEvent()メソッドにタッチイベントが渡されます。タッチイベントの種類は次の4つです。

Display.TOUCH_PRESSED_EVENT	タッチ
Display.TOUCH_MOVEDSTART_EVENT	ムーブスタート
Display.TOUCH_MOVEDEND_EVENT	ムーブエンド
Display.TOUCH_RELEASED_EVENT	リリース

processEvent()メソッドをオーバーライドして、引数*type*を調べて各種処理を行います。

［Canvasクラス］

void processEvent(*type, param***)**

［解説］　キャンバスのイベント処理
［引数］　*type*　イベント種別：int型
　　　　param　パラメータ：int型

7-4 待ち受けiアプリ

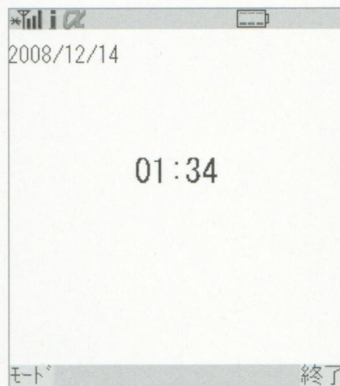


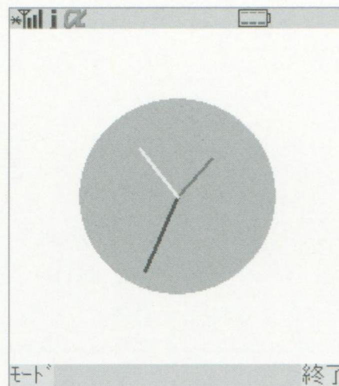
DoJa

http://book.mycom.co.jp/support/pc/star/d/74d.html

＊ **DoJaプロファイルの基本API**
（Starプロファイル端末では不可。但し現在のStar対応端末はDoJaと両対応なため動作可能）

時刻を表示する**待ち受けiアプリ**を作ります。この機能はDoJaプロファイルの基本APIです。Starプロファイルでは利用できません。ユーザが端末設定で「待ち受けアプリ-設定する」を指定することで待ち受けiアプリとして起動できます。





ソフトキー 1	デジタル表示とアナログ表示の切り替え
ソフトキー 2	活性化モードから非活性化モードへ

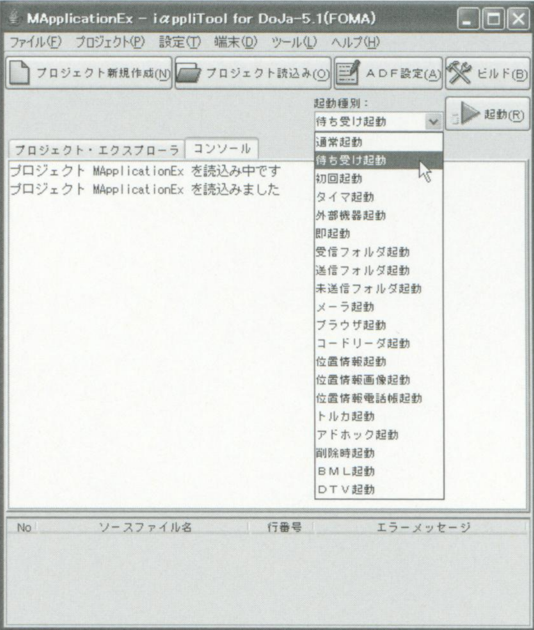
待ち受けiアプリは、待ち受けイメージのように携帯に常駐するiアプリのことです。実行中に電話がかかってきた時には一時停止しますが、通話が終わると再開します。

このプログラムは、次の2つのクラスで構成されています。

・**MApplicationExクラス（MApplicationEx.java）**
・**MApplicationCanvasクラス（MApplicationCanvas.java）**

プロジェクト名「**MApplicationEx**」でプロジェクトを作成してください。

エミュレータで「待ち受けiアプリ」として起動するには、iαppliToolのリストボックスで「待ち受け起動」を選択してから、[起動]ボタンをクリックします。



▶待ち受けiアプリの状態

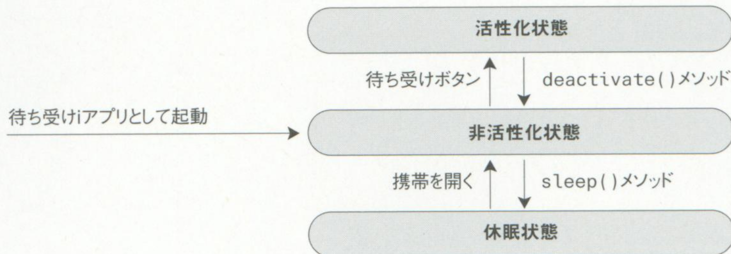
待ち受けiアプリは**活性化状態**、**非活性化状態**、**休眠状態**の3つの状態を遷移します。

活性化状態	通常のiアプリ実行と同じように、キーイベントを受信できる状態
非活性化状態	アニメーション表示だけ行う状態で、キーイベントはiアプリでなくネイティブOSが受信。イメージを待ち受けにしている時と同じように、電話をかけたりメールを送ったりできる
休眠状態	静止画表示だけを行う状態で、キーイベントはネイティブOSが受信。電池の消費を抑えることができる

待ち受けiアプリの起動時の状態は「非活性化状態」となります。

「活性化状態」から「非活性化状態」へ、「非活性化状態」から「休眠状態」へはiアプリからの命令で遷移させます。

携帯を開いた時に「休眠状態」から「非活性化状態」へ、[待ち受け]ボタンを押した時に「非活性化状態」から「活性化状態」へ遷移します。[待ち受け]ボタンがどのボタンなのかは機種依存となります。



▶MApplicationExクラス

MApplicationExクラスは、プログラムの本体となるクラスです。

DoJa MApplicationEx.java

```
import com.nttdocomo.ui.MApplication;
import com.nttdocomo.ui.Display;

// 待ち受けアプリ（本体）
public class MApplicationEx extends MApplication { // …❶

    // アプリの開始
    public void start() {
        MApplicationCanvas c = new MApplicationCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }

    // システムイベントの処理 …❷
    public void processSystemEvent(int type,int param) {
        // 休眠状態への遷移へ …❸
        if (type==FOLD_CHANGED_EVENT && param==0) {
            deactivate();
            sleep();
        }
    }

    // 終了
    void exit() {
        // 非活性化状態への遷移
        if (getLaunchType()==LAUNCHED_AS_CONCIERGE) { // …❹
```

Next page. ⬇

```
            deactivate();
        }
          // アプリの終了
        else {
            terminate();
        }
    }
}
```

MApplicationEx ...❶：MApplicationクラス

待ち受けiアプリの本体となるクラスは、IApplicationクラスでなく**MApplicationクラス**を継承する必要があります。

MApplicationEx ...❷：システムイベントの処理

待ち受けiアプリに関連するイベントが発生した時、実機からprocessSystemEvent()メソッドが呼ばれます。オーバーライドして必要な処理を記述してください。

DoJa ［MApplicationクラス］

void processSystemEvent(*type, param*)

［解説］ システムイベント発生時に呼ばれる
［引数］ *type* 待ち受けイベント種別：int型
Param パラメータ：int型

引数の*type*（待ち受けイベント種別）には次の定数が渡されます。パラメータは種別によって異なります。

MApplication.MODE_CHANGED_EVENT	活性化状態に切り替わった時に発生
MApplication.CLOCK_TICK_EVENT	時計の時刻が変化した時に発生
MApplication.FOLD_CHANGED_EVENT	折りたたみ状態が変化した時に発生
MApplication.WAKEUP_TIMER_EVENT	iアプリが設定した時間だけ経過した時に発生

MApplication.MODE_CHANGED_EVENTは活性化状態に切り替わった時に発生するイベントです。

MApplication.CLOCK_TICK_EVENTはネイティブ時計の時刻が変化した時（分単位）に発生するイベントです。setClockTick()メソッドでtrueを指定した時だけ発生します。パラメータには0時0分0秒からの経過時間を秒単位で表した値が渡されます。

1時23分45秒の場合、「1 * 3600 + 23 * 60 + 45 = 5025」が渡されます。

DoJa ［MApplicationクラス］

void setClockTick(*flag*)

［解説］ クロックイベントを発生させるかの指定

［引数］ *flag* クロックイベントを発生させるか：boolean型

MApplication.FOLD_CHANGED_EVENTは、折りたたみ状態が変化した時に発生するイベントです。パラメータには、閉じた時は0、開いた時は1が渡されます。

MApplication.WAKEUP_TIMER_EVENTはiアプリが設定した時間だけ経過した時に発生するイベントです。setWakeupTimer()メソッドで指定した時間が経過した時に発生します。タイマの残り時間を調べるには、getWakeupTimer()メソッド、タイマを解除するにはresetWakeupTimer()メソッドを使います。

DoJa ［MApplicationクラス］

void setWakeupTimer(*time*)

［解説］ ウェイクアップイベントを発生するまでの時間の指定

［引数］ *time* ウェイクアップイベントを発生するまでの時間（ミリ秒単位）：int型

DoJa ［MApplicationクラス］

int getWakeupTimer()

［解説］ ウェイクアップイベントを発生するまでの時間の取得

［戻り値］ ウェイクアップイベントを発生するまでの時間

DoJa ［MApplicationクラス］

void resetWakeupTimer()

［解説］ ウェイクアップイベントの解除

chapter

1

2

3

4

5

6

7

8

a

MApplicationEx ...❸：休眠状態への遷移

活性化状態から非活性化状態に遷移するには、MApplicationクラスのdeactivate()メソッドを使います。

DoJa ［MApplicationクラス］

void deactivate()

［解説］ 活性化状態から非活性化状態への遷移

非活性化状態から休眠状態に遷移するには、MApplicationクラスのsleep()メソッドを使います。

DoJa ［MApplicationクラス］

void sleep()

［解説］ 非活性化状態から休眠状態への遷移

活性化状態から休眠状態に直接遷移することはできません。

MApplicationEx ...❹：アプリケーションの起動形態

待ち受けiアプリはソフトメニューから「通常iアプリ」として起動することもできます。通常iアプリとして起動すると、非活性化状態や休眠状態へ遷移することはできません。

「待ち受けiアプリ」として起動したか、「通常iアプリ」として起動したかを知りたい時は、**IApplicationクラス**のgetLaunchType()メソッドを使います。

DoJa ［IApplicationクラス］

int getLaunchType()

［解説］ アプリケーションの起動形態の取得

［戻り値］ アプリケーションの起動形態

戻り値として次の定数が返されます。

IApplication.LAUNCHED_AFTER_DOWNLOAD	ダウンロード直後の起動
IApplication.LAUNCHED_AS_CONCIERGE	待ち受けiアプリとしての起動
IApplication.LAUNCHED_FROM_BROWSER	ブラウザからの起動
IApplication.LAUNCHED_FROM_EXT	外部インタフェース（赤外線通信等）からの起動
IApplication.LAUNCHED_FROM_MAILER	メーラからの起動
IApplication.LAUNCHED_FROM_MENU	ソフトメニューからの起動

IApplication.LAUNCHED_FROM_TIMER	タイマによる起動

今回は「待ち受けiアプリ」として起動してる時は「非活性化状態」へ遷移し、「通常iアプリ」として起動してる時はiアプリを終了させています。

▶MApplicationCanvasクラス

MApplicationCanvasクラスはキャンバスとなるクラスです。

DoJa MApplicationCanvas.java

```
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.IApplication;
import java.util.Calendar;

// 待ち受けiアプリ(キャンバス)
public class MApplicationCanvas extends Canvas
    implements Runnable {
    // 回転定数
    private final static int DX[] = {
        0,426,848,1261,1666,2046,2405,2737,3043,3312,
        3545,3739,3895,4006,4073,4096,4074,4007,3897,3742,
        3548,3316,3048,2741,2410,2052,1671,1267,854,433,
        0,-426,-848,-1261,-1666,-2046,-2405,-2737,-3043,-3312,
        -3545,-3739,-3895,-4006,-4073,-4096,-4074,-4007,-3897,-3742,
        -3548,-3316,-3048,-2741,-2410,-2052,-1671,-1267,-854,-433};
    private final static int DY[] = {
        -4096,-4074,-4007,-3897,-3742,-3548,-3316,-3048,-2741,-2410,
        -2052,-1671,-1267,-854,-433,0,426,848,1261,1666,
        2046,2405,2737,3043,3312,3545,3739,3895,4006,4073,
        4096,4074,4007,3897,3742,3548,3316,3048,2741,2410,
        2052,1671,1267,854,433,0,-426,-848,-1261,-1666,
        -2046,-2405,-2737,-3043,-3312,-3545,-3739,-3895,-4006,-4073};

    // 変数
    private int keyEvent = -999; // キーイベント
    private Graphics g;          // グラフィックス

    // 処理
    public void run() {
```

Next page.

```
        g = getGraphics();

        // 変数
        Calendar cal; // カレンダー
        int year;     // 年
        int month;    // 月
        int day;      // 日
        int hour;     // 時
        int minute;   // 分
        int second;   // 秒
        int mode = 0; // モード(0:ﾃﾞｼﾞﾀﾙ,1:ｱﾅﾛｸﾞ)

        // ソフトキー
        setSoftLabel(SOFT_KEY_1,"ﾓｰﾄﾞ");
        setSoftLabel(SOFT_KEY_2,"終了");

        while (true) {
            // 時刻の取得
            cal    = Calendar.getInstance();
            year   = cal.get(Calendar.YEAR);
            month  = cal.get(Calendar.MONTH)+1;
            day    = cal.get(Calendar.DAY_OF_MONTH);
            hour   = cal.get(Calendar.HOUR_OF_DAY);
            minute = cal.get(Calendar.MINUTE);
            second = cal.get(Calendar.SECOND);

            // 描画
            g.lock();
            g.setColor(g.getColorOfName(g.WHITE));
            g.fillRect(0,0,getWidth(),getHeight());

            // デジタル
            if (mode==0) {
                g.setColor(g.getColorOfName(g.BLACK));
                g.setFont(Font.getFont(Font.SIZE_SMALL));
                g.drawString(year       +"/"+
                    formatNum(month," ")+"/"+
                    formatNum(day," "),0,24);
                g.setFont(Font.getFont(Font.SIZE_MEDIUM));
                g.drawString(
                    formatNum(hour,  "0")+":"+
                    formatNum(minute,"0"),90,110);
            }

            // アナログ
            else {
                g.setColor(g.getColorOfName(g.SILVER));
                g.fillArc(50,50,140,140,0,360);
```

```
                g.setColor(g.getColorOfName(g.RED));
                drawLine(120,120,
                    120+DX[hour%12*5+minute/12]/110,
                    120+DY[hour%12*5+minute/12]/110);
                g.setColor(g.getColorOfName(g.BLUE));
                drawLine(120,120,120+DX[minute]/70,120+DY[minute]/70);
                g.setColor(g.getColorOfName(g.YELLOW));
                drawLine(120,120,120+DX[second]/120,120+DY[second]/70);
            }

            // アンロック
            g.unlock(true);

            // キーイベント
            if (keyEvent==Display.KEY_SOFT1) {
                mode = (mode+1)%2;
            } else if (keyEvent==Display.KEY_SOFT2) {
                ((MApplicationEx)IApplication.getCurrentApp()).exit();
            }
            keyEvent = -999;

            // スリープ
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    // ラインの描画
    private void drawLine(int x0,int y0,int x1,int y1) {
        g.drawLine(x0,y0,x1,y1);
        g.drawLine(x0+1,y0,x1+1,y1);
        g.drawLine(x0-1,y0,x1-1,y1);
        g.drawLine(x0,y0+1,x1,y1+1);
        g.drawLine(x0,y0-1,x1,y1-1);
    }

    // 数字フォーマットを整える
    private String formatNum(int num,String ume) {
        if (num<10) {
            return ume+num;
        } else {
            return ""+num;
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
```

Next page. ⬇

```
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

▶ADFの設定

ADFは「MyConcierge」を次のように設定してください。

MyConcierge	Yes

「MyConcierge」に「Yes」を指定して「待ち受けiアプリ」としての利用を許可します。この設定を行わないと、待ち受けiアプリとして起動することができません。

ゲームの作成

8

chapter

1
2
3
4
5
6
7
8
a

本章では、これまで解説してきた機能を利用して、3本のゲームを作成します。
これらのソースコードを参考に、自分だけのオリジナルゲームの作成に挑戦してみてください。

8-1 写真パズルゲーム



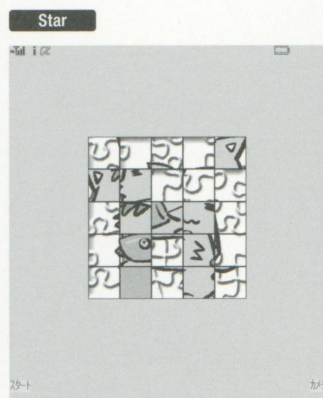
Star
http://book.mycom.co.jp/support/pc/star/81s.html



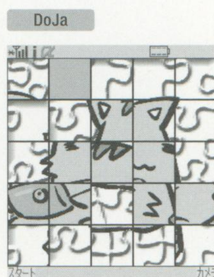
DoJa
http://book.mycom.co.jp/support/pc/star/d/81d.html

写真パズルゲームを作ります。5x5の25ピースに分割してバラバラになった絵を、元の1枚の絵に戻すゲームです。ソフトキー 1で絵がバラバラになります。方向キーでピースを移動させ、絵が元通りになればクリアです。ソフトキー 2でカメラモードに遷移し、そこで撮影した写真をパズルの絵柄として使うことができます。

Star



DoJa



方向キー	ピースの移動
ソフトキー 1	ゲーム開始
ソフトキー 2	写真撮影

今回のプログラムは、次の2つのクラスで構成されています。

- **PhotoPuzzle**クラス（**PhotoPuzzle.java**）
- **PhotoCanvas**クラス（**PhotoCanvas.java**）

プロジェクト名「**PhotoPuzzle**」でプロジェクトを作成してください。

▶イメージファイルの準備

今回使用するイメージファイルは次の1枚です。プロジェクトの「res」フォルダに置いてください。

・サンプルイメージ：**sample.jpg**（**240x240**ドット）



▶PhotoPuzzleクラス

PhotoPuzzleクラスは、プログラムの本体となるクラスです。

Star PhotoPuzzle.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;       //

// 写真パズルゲーム（本体）
public class PhotoPuzzle extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) {          // ⓒ
        Display.setCurrent(new PhotoCanvas());
    }
}
```

DoJa PhotoPuzzle.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class PhotoPuzzle extends IApplication {
```

ⓒ

```
    public void start() {
```

▶PhotoCanvasクラス

PhotoCanvasクラスはキャンバスとなるクラスです。

Star PhotoCanvas.java

```
import com.docomostar.device.Camera;       // ⓐ
import com.docomostar.media.MediaImage;    //
import com.docomostar.media.MediaManager;  //
import com.docomostar.ui.Canvas;           //
import com.docomostar.ui.Display;          //
import com.docomostar.ui.Graphics;         //
import com.docomostar.ui.Image;            //
import java.util.Random;                   //

// 写真パズルゲーム(キャンバス)
public class PhotoCanvas extends Canvas {
    // システム
    private Random rand = new Random(); // 乱数
    private Camera cam;                 // カメラ
    private Image  image;               // イメージ

    // ピースイメージとピースデータ …❶
    private Image[] piece = new Image[25];  // ピースイメージ
    private int[]   data = new int[25];     // ピースデータ

    // コンストラクタ
    public PhotoCanvas() {
        // ソフトキーの指定
        setSoftLabel(SOFT_KEY_1,"スタート"); // ⓑ
        setSoftLabel(SOFT_KEY_2,"カメラ");   //

        // サンプルイメージの読み込み
        try {
            MediaImage m = MediaManager.getImage(
                "resource:///sample.jpg");
            m.use();
            image = m.getImage();
        } catch (Exception e) {
```

Next page.

```
        }

        // ピースの生成
        makePiece(image);

        // カメラの初期化
        cam = Camera.getCamera(0);
        cam.setImageSize(240,240);

        // 描画
        repaint();
    }

    // ピースの生成 ...❷
    private void makePiece(Image image) {
        Graphics g;
        for (int i=0;i<25;i++) {
            if (piece[i]==null) piece[i] = Image.createImage(48,48);
            data[i] = i;
            g = piece[i].getGraphics();
            g.drawImage(image,-48*(i%5),-48*(i/5));
        }
    }

    // ピースの移動 ...❸
    private void movePiece(int key) {
        // 空きピースインデックスの検索
        int freeIdx = 0;
        for (;freeIdx<25;freeIdx++) {
            if (data[freeIdx]==24) break;
        }

        // ピースの交換
        if (key==Display.KEY_UP && freeIdx/5<4) {
            data[freeIdx] = data[freeIdx+5];
            data[freeIdx+5] = 24;
        } else if (key==Display.KEY_DOWN && freeIdx/5>0) {
            data[freeIdx] = data[freeIdx-5];
            data[freeIdx-5] = 24;
        } else if (key==Display.KEY_LEFT && freeIdx%5<4) {
            data[freeIdx] = data[freeIdx+1];
            data[freeIdx+1] = 24;
        } else if (key==Display.KEY_RIGHT && freeIdx%5>0) {
            data[freeIdx] = data[freeIdx-1];
            data[freeIdx-1] = 24;
        }
    }

    // ピースの描画 ...❹
```

```
    public void paint(Graphics g) {
        int i;
        if (image==null) return;

        // 結果
        boolean complete = true;
        for (i=0;i<25;i++) {
            if (data[i] != i) complete = false;
        }

        // 描画
        g.lock();
        g.setColor(g.getColorOfName(g.SILVER));
        g.fillRect(0,0,240,240);
        g.setColor(g.getColorOfName(g.BLACK));
        for (i=0;i<25;i++) {
            if (complete || data[i] != 24) {
                g.drawImage(piece[data[i]],48*(i%5),48*(i/5));
            }
            if (!complete) {
                g.drawRect(48*(i%5),48*(i/5),47,47);
            }
        }
        g.unlock(true);
    }

    // キーイベントの処理
    public void processEvent(int type,int param) {
        if (type==Display.KEY_PRESSED_EVENT) {
            // ゲーム開始
            if (param==Display.KEY_SOFT1) {
                for (int i=99;i >= 0;i--) {
                    movePiece(Display.KEY_LEFT+(rand.nextInt()>>>1)%4);
                }
            }
            // 写真撮影
            else if (param==Display.KEY_SOFT2) {
                try {
                    cam.takePicture();
                    if (cam.getNumberOfImages() != 0) {
                        MediaImage m = (MediaImage)cam.getImage(0); // ⓒ
                        m.use();
                        Image photo = m.getImage();
                        makePiece(photo);
                        photo.dispose();
                    }
                } catch (Exception e) {
                }
            }
```

Next page. ⬇

```
                // ピースの移動
                else {
                    movePiece(param);
                }
                repaint();
            }
        }
    }
```

DoJa PhotoCanvas.java

ⓐ

```
import com.nttdocomo.device.Camera;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.MediaManager;
import java.util.Random;
```

ⓑ

```
        // ソフトキーの指定
        setSoftLabel(SOFT_KEY_1,"ｽﾀｰﾄ");
        setSoftLabel(SOFT_KEY_2,"ｶﾒﾗ");
```

ⓒ

```
                    MediaImage m = cam.getImage(0);
```

PhotoCanvas …❶：ピースイメージとピースデータ

今回作るゲームは5x5の25ピースのパズルです。ピースイメージは、piece配列に格納します。ピースがどのように並んでいるかというピースデータは、data配列に格納します。

X座標は「インデックス%5」、Y座標は「インデックス/5」で計算します。

data配列のインデックスはピースの位置、data配列の値はピースの絵柄に対応します。data[0]の値が「4」の時、絵柄の右上のピースが、現在左上にあることを示します。また、data配列の値が「24」（右下のピース）の時は、そこを空き領域として周囲のピースを詰めることができます。

PhotoCanvas ...❷：ピースの生成

ピースの生成はmakePiece()メソッドで行っています。240x240の画像を48x48の画像にずらして描画することにより、25個の小さなピースイメージに分割しています。

［PhotoCanvasクラス］

void makePiece(*image*)

［解説］ ピースの生成
［引数］ *image* イメージ：Image型

また、動的なImageオブジェクトの生成はImageクラスのcreateImage()メソッドで行います。

［PhotoCanvasクラス］

static Image createImage(*width, height*)

［解説］ 動的なImageオブジェクトの生成
［引数］ *width* 幅：int型
height 高さ：int型
［戻り値］ Imageオブジェクト

PhotoCanvas ...❸：ピースの移動

ピースの移動はmovePiece()メソッドで行っています。方向キーによるピースの移動や、ソフトキー1でバラバラにする時に使います。

［PhotoCanvasクラス］

void movePiece(*key*)

［解説］ ピースの移動
［引数］ *key* キー：int型

空ピース（data配列の値が24）を検索し、上キーが押された時は空ピースとその下のピース、下キーが押された時は空ピースとその上のピース、左キーが押された時は空ピースとその右のピース、右キーが押された時は空ピースとその左のピースを交換します。

PhotoCanvas …❹：ピースの描画

ピースの描画はpaint()メソッドで行います。描画する前にパズルが完成しているかどうかを調べます。配列のインデックスと中身が全て同じ時、絵は完成したことになります。また、パズルが完成していない時は、空ピースを空き領域として描画しないようにします。

▶ADFの設定

ADFはAppNameとDrawAreaを次のように設定してください。

AppName	写真パズル
DrawArea	240x240

8-2 アクションゲーム



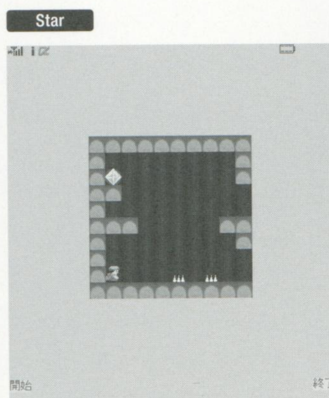
Star
http://book.mycom.co.jp/support/pc/star/82s.html



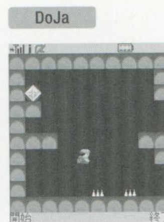
DoJa
http://book.mycom.co.jp/support/pc/star/d/82d.html

アクションゲーム「**トレジャーゲッター**」を作ります。プレイヤーは少年を操作して、炭鉱に埋もれている宝石を探します。少年は左右キーで移動し、上キーか［#］キーでジャンプします。宝石をゲットできればクリア、針の罠にかかってしまうとゲームオーバーになります。

Star



DoJa



左キー	左へ移動
右キー	右へ移動
上キー・[#]キー	ジャンプ
ソフトキー1	ゲーム開始
ソフトキー2	ゲーム終了

今回のプログラムは、次の2つのクラスで構成されています。

- **ActionGameクラス（ActionGame.java）**
- **ActionCanvasクラス（ActionCanvas.java）**

プロジェクト名「**ActionGame**」でプロジェクトを作成してください。

▶イメージファイルの用意

今回使用するイメージファイルは次の5枚です。プロジェクトの「res」フォルダに置いてください。

背景：0.gif（24x24ドット）	壁：1.gif（24x24ドット）	針の罠：2.gif（24x24ドット）
宝石：3.gif（24x24ドット）	少年：4.gif（24x24ドット）	

▶サウンドファイルの用意

今回使用するサウンドファイルは次の3つです。プロジェクトの「res」フォルダに置いてください。

・プレイ時のBGM：0.mld
・クリア時のBGM：1.mld
・ゲームオーバー時のBGM：2.mld

▶テキストファイルの用意

今回のゲームのマップデータは、テキストファイルとしてJARファイルに入れます。次のテキストファイルをエディタで作成して、プロジェクトの「res」フォルダに置いてください。マップチップの数値の意味は次の通りです。

0:　背景　　1:　壁
2:　針の罠　3:　宝石

・テキストファイル：**map.txt**

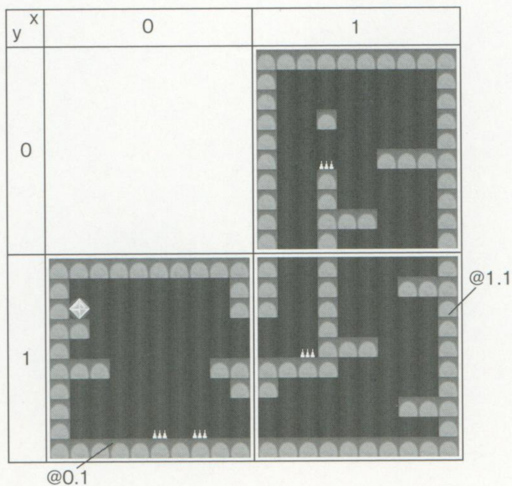
```
@1,0
1111111111
1000000001
1000000001
1001000001
1000000001
1002001111
1001000001
```

```
1001000001
1001110001
1001000001

@0,1
1111111111
1000000001
1300000001
1100000000
1000000000
1110000011
1000000001
1000000000
1000020200
1111111111

@1,1
1001000001
1001000111
1001000001
0001000001
0021110001
1111000001
1000000001
0000000111
0000000001
1111111111
```

このテキストは3つ分のマップデータを保持しています。マップデータの先頭で @X座標,Y座標 の書式で座標を記述し、その次の行から 10x10 のマップチップ情報を記述しています。



chapter
1
2
3
4
5
6
7
8
a

▶ActionGameクラス

ActionGameクラスは、プログラムの本体となるクラスです。

Star ActionGame.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// アクションゲーム(本体)
public class ActionGame extends StarApplication { // ⓑ
    private ActionCanvas c;

    // アプリの開始
    public void started(int launchType) {         // ⓒ
        c = new ActionCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }

    // サウンドの再開 ...❶
    public void activated(int activateInfo) {                 // ⓓ
        if (activateInfo != ACTIVATED_FROM_STARTED_STATE) { //
            c.resumeSound();                                //
        }                                                   //
    }
}
```

DoJa ActionGame.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class ActionGame extends IApplication {
```

ⓒ

```
    public void start() {
```

ⓓ

```
    public void resume() {
        c.resumeSound();
    }
```

ActionGame ...❶：サウンドの再開

iアプリは電話の着信などで一時停止するとサウンドは停止してしまいます。そこでiアプリ復帰時に呼ばれるメソッドをオーバーライドし、サウンドの再開処理を行っています。

Star

iアプリの開始時および再開時にはStarApplicationクラスのactivated()メソッドが呼ばれます。

Star ［StarApplicationクラス］

void activated(*activateInfo*)

［解説］ アプリの開始時と再開時に呼ばれる
［引数］ *activateInfo* アクティブ情報:int型

アクティブ情報にはiアプリ起動時にはACTIVATED_FROM_STARTED_STATE、それ以外の時は一時停止理由の定数が渡されます。

StarApplication.ACTIVATED_BY_KEY_TOUCHED	ユーザのキータッチによる活性化
StarApplication.ACTIVATED_BY_NATIVE	ネイティブ機能による活性化
StarApplication.ACTIVATED_BY_WAKEUP_TIMER	起床タイマによる活性化
StarApplication.ACTIVATED_BY_WAKEUP_TIMER_DELAYED	起床タイマによる活性化（遅延あり）
StarApplication.ACTIVATED_FROM_STARTED_STATE	started(int)の実行完了による活性化

DoJa

iアプリの再開時にはIApplicationクラスのresume()メソッドが呼ばれます。

DoJa ［IApplicationクラス］

void resume()

［解説］ iアプリの再開時に呼ばれる

▶ActionCanvasクラス

ActionCanvasクラスは、キャンバスとなるクラスです。

Star ActionCanvas.java

```
import com.docomostar.StarApplication;    // ⓐ
import com.docomostar.media.MediaImage;   //
import com.docomostar.media.MediaManager; //
```

Next page.

```
import com.docomostar.media.MediaSound;    //
import com.docomostar.ui.AudioPresenter;   //
import com.docomostar.ui.Canvas;           //
import com.docomostar.ui.Display;          //
import com.docomostar.ui.Font;             //
import com.docomostar.ui.Graphics;         //
import com.docomostar.ui.Image;            //
import com.docomostar.ui.MediaListener;    //
import com.docomostar.ui.MediaPresenter;   //
import javax.microedition.io.Connector;    //
import java.io.InputStream;

// アクションゲーム(キャンバス)
public class ActionCanvas extends Canvas
    implements Runnable,MediaListener {
    // シーン定数
    private final static int S_PLAY     = 0; // プレイ
    private final static int S_CLEAR    = 1; // クリア
    private final static int S_GAMEOVER = 2; // ゲームオーバー

    // システム
    private int      scene    = S_PLAY;        // シーン …❶
    private int      init     = S_PLAY;        // 初期化
    private int      keyEvent = -999;          // キーイベント
    private Graphics g        = getGraphics(); // グラフィックス
    private Image[]  image    = new Image[5];  // イメージ

    // マップ
    private int     mapX;                       // X座標
    private int     mapY;                       // Y座標
    private int[][] mapData = new int[10][10]; // データ

    // 少年
    private int boyX;    // X座標
    private int boyY;    // Y座標
    private int boyDir;  // 方向(0:左,1:右)
    private int boyJump; // ジャンプ(-1:着地,0:下降,1以上:上昇)

    // サウンド
    private AudioPresenter player;                     // プレイヤー
    private MediaSound[] sound = new MediaSound[3]; // サウンド

    public void run() {
        int i,j,key;
        long sleepTime = 0L;

        // 起動時の初期化
        try {
            // イメージの読み込み
```

```
        MediaImage m;
        for (i=0;i<5;i++) {
            m = MediaManager.getImage("resource:///"+i+".gif");
            m.use();
            image[i] = m.getImage();
        }

        // サウンドの読み込み
        for (i=0;i<3;i++) {
            sound[i] = MediaManager.getSound("resource:///"+i+".mld");
            sound[i].use();
        }
    } catch (Exception e) {
    }

    // プレイヤーの初期化
    player = AudioPresenter.getAudioPresenter();
    player.setMediaListener(this);

    // ソフトラベル
    setSoftLabel(SOFT_KEY_1,"開始");
    setSoftLabel(SOFT_KEY_2,"終了");

    // 無限ループ
    while (true) {
        // シーンの初期化
        if (init >= 0) {
            // プレイ
            if (init==S_PLAY) {
                // マップ
                mapX = 0;
                mapY = 1;
                loadMap();

                // 少年
                boyX    = 36;
                boyY    = 216;
                boyDir  = 1;
                boyJump = -1;

                // サウンド
                playSound(0);
            }
            // クリア
            if (init==S_CLEAR) {
                // サウンド
                playSound(1);
            }
            // ゲームオーバー
```

Next page. ⬇

```
        if (init==S_GAMEOVER) {
            // サウンド
            playSound(2);
        }
        // 共通
        scene   =init;
        init     = -1;
        keyEvent = -999;
    }

    // マップの描画
    g.lock();
    for (j=0;j<10;j++) {
        for (i=0;i<10;i++) {
            g.drawImage(image[mapData[j][i]],24*i,24*j);
        }
    }

    // 少年の描画
    if (boyDir != 0) g.setFlipMode(g.FLIP_HORIZONTAL);
    g.drawImage(image[4],boyX-12,boyY-24);
    if (boyDir != 0) g.setFlipMode(g.FLIP_NONE);

    // 文字列の描画
    g.setFont(Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,24)); // ⓑ
    if (scene==S_CLEAR)    drawBold("クリア！",72,120);
    if (scene==S_GAMEOVER) drawBold("ゲームオーバー",36,120);
    g.unlock(true);

    // ソフトキーの処理
    if (keyEvent==Display.KEY_SOFT1) init = S_PLAY;
    if (keyEvent==Display.KEY_SOFT2) {                          // ⓒ
        StarApplication.getThisStarApplication().terminate(); //
    }                                                           //

    // プレイの処理
    if (scene==S_PLAY) {
        key = getKeypadState();
        // 左移動
        if (((1<<Display.KEY_LEFT)&key) != 0) {
            boyDir = 0;
            // マップ
            if (boyX-19<0) {
                mapX--;loadMap();
                boyX = 240;
            }
            // 少年
            else if (getHit(-19,-20) != 1 && getHit(-19,-4) != 1) {
                boyX -= 8;
```

```
        }
    }
    // 右移動
    if (((1<<Display.KEY_RIGHT)&key) != 0) {
        boyDir = 1;
        // マップ
        if (boyX+19 >= 240) {
            mapX++;loadMap();
            boyX = 0;
        }
        // 少年
        else if (getHit(19,-20) != 1 && getHit(19,-4) != 1) {
            boyX += 8;
        }
    }
    // ジャンプ終了
    if ((((1<<Display.KEY_UP)   &key)==0 &&
         ((1<<Display.KEY_POUND)&key)==0) && boyJump>0) boyJump = 0;
    // ジャンプ開始
    if ((keyEvent==Display.KEY_UP ||
         keyEvent==Display.KEY_POUND) && boyJump==-1) boyJump = 11;
    keyEvent = -999;

    // 上昇
    if (boyJump>1) {
        boyJump--;
        boyY -= 8;
        // マップ
        if (boyY-24<0) {
            mapY--;loadMap();
            boyY = 240;
        }
        // 少年
        else if (getHit(8,-24)==1 || getHit(-8,-24)==1) {
            boyY += (24-boyY%24);
            boyJump = 0;
        }
    }
    // 下降
    else {
        boyY += 8;
        // マップ
        if (boyY >= 240) {
            mapY++;loadMap();
            boyY = 0;
        }
        // 少年
        else if (getHit(8,0)==1 || getHit(-8,0)==1) {
            boyY -= boyY%24;
```

Next page.

```
                    boyJump = -1;
                    // ゲームオーバー
                    if (getHit(8,-24)==2 || getHit(-8,-24)==2) init = S_GAMEOVER;
                    // クリア
                    if (getHit(8,-24)==3 || getHit(-8,-24)==3) init = S_CLEAR;
                } else {
                    boyJump = 0;
                }
            }
        }

        // スリープ
        while (System.currentTimeMillis()<sleepTime+100L);
        sleepTime = System.currentTimeMillis();
    }
}

// 少年の衝突判定
private int getHit(int x,int y) {
    x += boyX;
    y += boyY;
    if (x<0 || 240<=x || y<0 || 240<=y) return 0;
    return mapData[y/24][x/24];
}

// サウンド再生 ...❷
private void playSound(int idx) {
    try {
        player.stop();
    } catch (Exception e) {
    }
    try {
        player.setSound(sound[idx]);
        player.play();
    } catch (Exception e) {
    }
}

// サウンド再開
public void resumeSound() {
    if (scene==S_PLAY) playSound(0);
}

// マップの読み込み
private void loadMap() {
    int i,j,k;
    String[] text = null;
    String key = "@"+mapX+","+mapY;
```

```
        // テキスト読み込み
        try {
            InputStream in = Connector.openInputStream(
                "resource:///map.txt");
            byte[] data = new byte[1024];
            int size = in.read(data);
            in.close();
            String str = new String(data,0,size);
            text = parseString(str,'¥n');
        } catch (Exception e) {
        }

        // 先頭検索
        for (k=0;k<text.length;k++) {
            if (text[k].equals(key)) break;
        }
        k++;

        // マップデータ生成
        for (j=0;j<10;j++) {
            for (i=0;i<10;i++) {
                mapData[j][i] = Integer.parseInt( // ...❺
                    text[j+k].substring(i,i+1));
            }
        }
    }

    // 文字列を任意の文字で分割 ....❹
    private String[] parseString(String str,char sep) {
        int i,j,size;
        String[] result;

        // ¥r を削除
        if (str.indexOf('¥r') >= 0) {
            StringBuffer sb = new StringBuffer();
            for (i=0;i<str.length();i++) {
                if (str.charAt(i) != '¥r') sb.append(str.charAt(i));
            }
            str = sb.toString();
        }

        // 最後尾に分割文字
        if (str.equals("")||str.charAt(str.length()-1) != sep) str += sep;

        // サイズを得る
        size = 0;
        i = str.indexOf(sep);
        while (i >= 0) {
            i = str.indexOf(sep,i+1);
```

Next page. ⬇

```
            size++;
        }

        // 分割する
        result = new String[size];
        size = 0;
        j = 0;
        i = str.indexOf(sep);
        while (i >= 0) {
            result[size++] = str.substring(j,i);
            j = i+1;
            i = str.indexOf(sep,j);
        }
        return result;
    }

    // 太文字の描画
    private void drawBold(String text,int x,int y) {
        g.setColor(g.getColorOfRGB(255,0,0));
        g.drawString(text,x-1,y-1);
        g.drawString(text,x  ,y-1);
        g.drawString(text,x+1,y-1);
        g.drawString(text,x-1,y);
        g.drawString(text,x+1,y);
        g.drawString(text,x-1,y+1);
        g.drawString(text,x  ,y+1);
        g.drawString(text,x+1,y+1);
        g.setColor(g.getColorOfRGB(255,120,120));
        g.drawString(text,x,y);
    }

    // メディアアクションのイベント処理
    public void mediaAction(MediaPresenter source,int type,int param) {
        // サウンドのループ ...❸
        if (type==AudioPresenter.AUDIO_COMPLETE && scene==S_PLAY) {
            try {
                player.play();
            } catch (Exception e) {
            }
        }
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
```

```
}
```

DoJa ActionCanvas.java

ⓐ

```
import com.nttdocomo.ui.AudioPresenter;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.MediaImage;
import com.nttdocomo.ui.MediaListener;
import com.nttdocomo.ui.MediaManager;
import com.nttdocomo.ui.MediaPresenter;
import com.nttdocomo.ui.MediaSound;
import javax.microedition.io.Connector;
import java.io.InputStream;
```

ⓑ

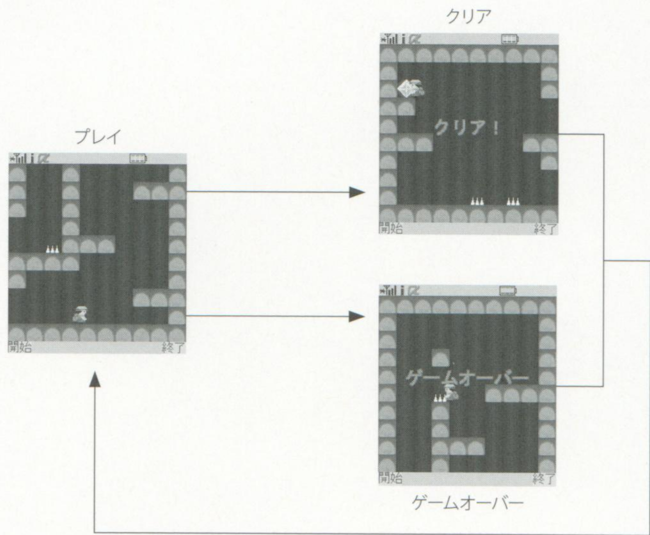
```
            g.setFont(Font.getFont(Font.SIZE_MEDIUM));
```

ⓒ

```
            if (keyEvent==Display.KEY_SOFT2)
                IApplication.getCurrentApp().terminate();
```

ActionCanvas ...❶：シーン

このゲームには次の3つの**シーン**があります。現在のシーンはscene変数で保持しています。次に遷移するシーンはinit変数で保持しています。遷移しない時は、initは「-1」を保持しています。



プレイ（S_PLAY）	ゲームをプレイするシーン。宝石をゲットした時は「クリア」に、針の罠にはまった時は「ゲームオーバー」に遷移
クリア（S_CLEAR）	"クリア！"という文字列を表示するシーン。ソフトキー1を押した時は「プレイ」に遷移
ゲームオーバー（S_GAMEOVER）	"ゲームオーバー"という文字列を表示するシーン。ソフトキー1を押した時は「プレイ」に遷移

ActionCanvas ...❷：サウンドの再生

サウンド番号を指定することでサウンドを再生するplaySound()メソッドを作ります。

［ActionCanvasクラス］

void playSound(*idx*)

［解説］ サウンドの再生
［引数］ *idx* サウンド番号：int型

stop()メソッドで再生中のサウンドを停止させ、setSound()メソッドで指定したMediaSoundオブジェクトをセットし、play()メソッドで再生します。

ActionCanvas ...❸：サウンドのループ

サウンドのループは、シーンが「プレイ」の時のみ行うようにしています。

ActionCanvas ...❹：文字列を任意の文字で分割

文字列を任意の文字で分割するparseString()メソッドを作ります。

［ActionCanvasクラス］

String[] parseString(*str, sep*)

［解説］ 文字列を任意の文字で分割
［引数］ *str* 分割前の文字列：char型
sep 分割文字：char型
［戻り値］ 分割後の文字列

はじめに、分割前の文字列の最後尾に**分割文字**があるかどうか調べ、ない時は分割文字を追加します。その後、分割文字の数を調べ、分割後の文字列の配列を作ります。最後にその配列に、分割した文字列をセットしてから戻します。

ActionCanvas ...❺：文字列の数値変換

文字列を数値に変換するには、IntegerクラスのparseInt()メソッドを使います。

［Integerクラス］

static int parseInt(*str*)

［解説］ 文字列の数値変換
［引数］ *str* 数値に変換する文字列：String型
［戻り値］ 変換後の数値

▶ADFの設定

ADFはAppNameとDrawAreaを次のように設定してください。

AppName	トレジャーゲッター
DrawArea	240x240

8-3 3Dレースゲーム



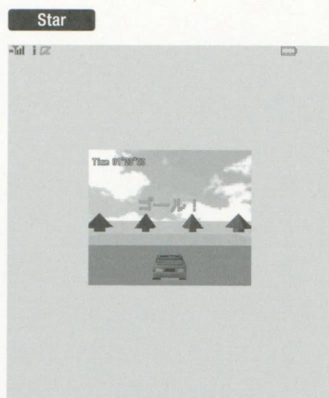
Star
http://book.mycom.co.jp/support/pc/star/83s.html



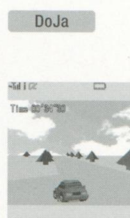
DoJa
http://book.mycom.co.jp/support/pc/star/d/83d.html

3Dレースゲーム「**ポリゴンレーシング**」を作ります。車を運転して、ゴールするまでにかかる時間を競うゲームです。[#]キーまたは [*]キーで前進し、左右キーで左右に向きを変えます。赤いラインがゴールとなります。

Star



DoJa



#キー、*キー	アクセル
0キー	ブレーキ
左キー	ハンドルを左にきる
右キー	ハンドルを右にきる
決定キー	ゲーム開始（ゴール時）

今回のプログラムは、次の3つのクラスで構成されています。

- **RaceGameクラス（RaceGame.java）**
- **RaceCanvasクラス（RaceCanvas.java）**
- **Mapクラス（Map.java）**

プロジェクト名「**RaceGame**」でプロジェクトを作成してください。

▶イメージファイルの用意

今回使用するイメージファイルは次の6枚です。プロジェクトの「res」フォルダに置いてください。

車 右端：0.gif（70x44ドット）	車 右：1.gif（70x44ドット）	車 前：2.gif（70x44ドット）
車 左：3.gif（70x44ドット）	車 左端：4.gif（70x44ドット）	背景：5.gif（240x120ドット）

▶RaceGameクラス

RaceGameクラスは、プログラムの本体となるクラスです。

Star RaceGame.java

```
import com.docomostar.StarApplication; // ⓐ
import com.docomostar.ui.Display;      //

// 3D レースゲーム ( 本体 )
public class RaceGame extends StarApplication { // ⓑ

    // アプリの開始
    public void started(int launchType) { // ⓒ
        RaceCanvas c = new RaceCanvas();
        Display.setCurrent(c);
        (new Thread(c)).start();
    }
}
```

DoJa RaceGame.java

ⓐ

```
import com.nttdocomo.ui.IApplication;
import com.nttdocomo.ui.Display;
```

ⓑ

```
public class RaceGame extends IApplication {
```

ⓒ

```
    public void start() {
```

▶RaceCanvasクラス

RaceCanvasクラスは、キャンバスとなるクラスです。

Star RaceCanvas.java

```
import com.docomostar.media.MediaImage;          // ⓐ
import com.docomostar.media.MediaManager;        //
import com.docomostar.ui.graphics3d.Graphics3D; //
import com.docomostar.ui.util3d.FastMath;        //
import com.docomostar.ui.util3d.Transform;       //
import com.docomostar.ui.util3d.Vector3D;        //
import com.docomostar.ui.Canvas;                 //
import com.docomostar.ui.Display;                //
import com.docomostar.ui.Font;                   //
import com.docomostar.ui.Graphics;               //
import com.docomostar.ui.Image;                  //
import javax.microedition.io.Connector;          //
import java.io.InputStream;                      //

// 3D レースゲーム ( キャンバス )
public class RaceCanvas extends Canvas
    implements Runnable {
    // シーン ...❶
    private final static int S_PLAY = 0; // プレイ
    private final static int S_GOAL = 1; // ゴール

    // システム
    private int      scene    = S_PLAY; // シーン
    private int      init     = S_PLAY; // 初期化
    private int      keyEvent = -999;   // キーイベント
    private int      time     = 0;      // 時間
    private Graphics g        = getGraphics(); // グラフィックス
    private Image[]  image    = new Image[6];  // イメージ

    // 車
    private int carX;      // X 座標
    private int carZ;      // Z 座標
    private int carDir;    // 方向
```

```
    private int carState; // 状態
    private int carSpeed; // 速さ

    // マップ
    private Map map = new Map();

    // 処理
    public void run() {
        int i,j,keyState;
        int dx,dz;
        long sleepTime = 0L;
        MediaImage m;

// ====================
// ゲームの初期化
// ====================
        try {
            for (i=0;i<6;i++) {
                m = MediaManager.getImage("resource:///"+i+".gif");
                m.use();
                image[i] = m.getImage();
            }
        } catch (Exception e) {
        }
        map.init(g);

        while (true) {
// ====================
// 初期化
// ====================
            if (init >= 0) {
                // プレイ
                if (init==S_PLAY) {
                    time     = 0;
                    carX     = 500*3;
                    carZ     = 500*160;
                    carDir   = 90;
                    carState = 2;
                    carSpeed = 0;
                }
                // 共通
                scene    = init;
                init     = -1;
                keyEvent = -999;
            }
            // ロック
            g.lock();
```

Next page.

```
// ====================
// 共通
// ====================
            // 背景の描画
            g.drawImage(image[5],0,0);
            g.setColor(g.getColorOfRGB(96,224,96));
            g.fillRect(0,105,240,95);

            // マップの描画
            map.draw(carX,carZ,carDir);

            // 車の描画
            g.drawImage(image[carState],85,150);

            // タイムの描画
            g.setFont(Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,12)); // ⓑ
            drawBold("Time "+formatNum(time/600,2)+"'"+
                formatNum((time/10)%60,2)+"¥""+(time%10)+"0",
                6,22,0);


// ====================
// プレイ
// ====================
            if (scene==S_PLAY) {
                // 加速・減速
                keyState = getKeypadState();
                if (((1<<Display.KEY_POUND)&keyState) != 0 ||
                    ((1<<Display.KEY_ASTERISK)&keyState) != 0) {
                    carSpeed += 10;
                    if (carSpeed>50) carSpeed = 50;
                } else if (((1<<Display.KEY_0)&keyState) != 0) {
                    carSpeed -= 10;
                    if (carSpeed<-20) carSpeed = -20;
                } else {
                    if (carSpeed>0) carSpeed -= 20;
                    if (carSpeed<0) carSpeed += 20;
                    if (Math.abs(carSpeed)<20) carSpeed = 0;
                }

                // 右折・左折
                if ((((1<<Display.KEY_LEFT )&keyState) != 0 && carSpeed>0) ||
                    (((1<<Display.KEY_RIGHT)&keyState) != 0 && carSpeed<0)) {
                    if (carState<4) carState++;
                    carDir -= 10;
                    if (carDir<0) carDir += 360;
                } else if ((((1<<Display.KEY_RIGHT)&keyState) != 0 && carSpeed>0) ||
                    (((1<<Display.KEY_LEFT)&keyState) != 0 && carSpeed<0)) {
                    if (carState>0) carState--;
```

```
                    carDir += 10;
                    if (carDir>360) carDir -= 360;
                } else {
                    if (carState>2) carState--;
                    if (carState<2) carState++;
                }

                // 衝突判定
                dx = carX-(int)(FastMath.cos(carDir)*300); // …❷
                dz = carZ-(int)(FastMath.sin(carDir)*300);
                if (map.getTip(dx,dz) >= 2) {
                    carSpeed = 0;
                } else if (map.getTip(dx,dz)==1) {
                    init = S_GOAL;
                }
                // 移動
                else {
                    dx = carX-(int)(FastMath.cos(carDir)*carSpeed*3);
                    dz = carZ-(int)(FastMath.sin(carDir)*carSpeed*3);
                    carX = dx;
                    carZ = dz;
                }

                // 時間経過
                time++;
            }

// ====================
// ゴール
// ====================
            if (scene==S_GOAL) {
                g.setFont(Font.getFont(Font.FACE_SYSTEM|Font.STYLE_PLAIN,24)); // ⓒ
                drawBold("ゴール！",(240-4*24)/2,90,1);
                if (keyEvent==Display.KEY_SELECT) init = S_PLAY;
            }

// ====================
// 後処理
// ====================
            // アンロック
            g.unlock(true);

            // スリープ
            while (System.currentTimeMillis()<sleepTime+60L);
            sleepTime = System.currentTimeMillis();
        }
    }

    // 数字フォーマットを整える
```

Next page. ⬇

```
    private String formatNum(int num,int len) {
        String str = ""+num;
        while (str.length()<len) str = "0"+str;
        return str;
    }

    // 太文字の描画(0:白,1:赤)
    private void drawBold(String text,int x,int y,int col) {
        if (col==0) g.setColor(g.getColorOfName(g.BLACK));
        if (col==1) g.setColor(g.getColorOfRGB(255,0,0));
        g.drawString(text,x-1,y-1);
        g.drawString(text,x  ,y-1);
        g.drawString(text,x+1,y-1);
        g.drawString(text,x-1,y);
        g.drawString(text,x+1,y);
        g.drawString(text,x-1,y+1);
        g.drawString(text,x  ,y+1);
        g.drawString(text,x+1,y+1);
        if (col==0) g.setColor(g.getColorOfName(g.WHITE));
        if (col==1) g.setColor(g.getColorOfRGB(255,120,120));
        g.drawString(text,x,y);
    }

    // キーイベントの処理
    public void processEvent(int type, int param) {
        if (type==Display.KEY_PRESSED_EVENT) keyEvent = param;
    }

    // 描画
    public void paint(Graphics g) {}
}
```

DoJa RaceCanvas.java

ⓐ

```
import com.nttdocomo.ui.graphics3d.*;
import com.nttdocomo.ui.util3d.*;
import com.nttdocomo.ui.Canvas;
import com.nttdocomo.ui.Display;
import com.nttdocomo.ui.Font;
import com.nttdocomo.ui.Graphics;
import com.nttdocomo.ui.Image;
import com.nttdocomo.ui.MediaManager;
import com.nttdocomo.ui.MediaImage;
import javax.microedition.io.*;
import java.io.*;
```

ⓑ

```
        // タイムの描画
        g.setFont(Font.getFont(Font.SIZE_SMALL));
```

ⓒ

```
            g.setFont(Font.getFont(Font.SIZE_MEDIUM));
```

RaceCanvas ...❶：シーン

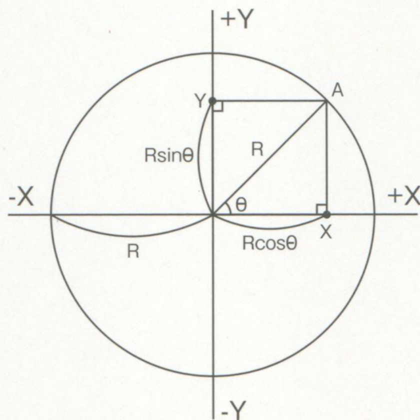
このゲームには次の2つの**シーン**があります。現在のシーンは`scene`変数で保持しています。次に遷移するシーンは`init`変数で保持しています。遷移しない時には、`init`は「`-1`」を保持しています。

プレイ（S_PLAY）	レースするシーン。ゴールした時には、シーンは「ゴール」に遷移する
ゴール（S_GOAL）	ゴールした時のシーン。決定ボタンを押した時には、シーンは「プレイ」に遷移する

RaceCanvas ...❷：数値演算ユーティリティ

`FastMath`クラスは**数値計算**のためのユーティリティクラスです。**三角関数**や**平方根**などの計算を行うことができます。

三角関数は、直角三角形の時に成り立つ、角度から位置を求める方法のことです。`sin`（サイン、正弦）と`cos`（コサイン、余弦）は、直角三角形の辺長の割合のことで、点Aの座標は、(Rcos θ，Rsin θ)となります。



［FastMathクラス］

static float sin(*a*)

［解説］ 正弦（sin、サイン）の計算
［引数］ *a* 角度（単位は度）：float型
［戻り値］ 正弦の近似値

［FastMathクラス］

static float cos(*a*)

［解説］ 余弦（cos、コサイン）の計算
［引数］ *a* 角度（単位は度）：float型
［戻り値］ 余弦の近似値

▶Mapクラス

Mapクラスは、3Dグラフィックスで表示するマップとなるクラスです。

Star Map.java

```
import com.docomostar.ui.graphics3d.Graphics3D; // ⓐ
import com.docomostar.ui.graphics3d.Light;      //
import com.docomostar.ui.graphics3d.Primitive;  //
import com.docomostar.ui.util3d.FastMath;       //
import com.docomostar.ui.util3d.Transform;      //
import com.docomostar.ui.util3d.Vector3D;       //
import com.docomostar.ui.Graphics;              //
import javax.microedition.io.Connector;         //

// 3D レースゲーム ( マップ )
public class Map {
    // マップデータ …❶
    private final static int[] MAP = { // マップ
        0,3,1,3,2,3,2,3,2,3,
        4,5,4,5,4,5,4,5,4,5,
        6,3,2,3,2,3,2,3,
        8,3,2,3,2,3,2,3,
        6,3,2,3,2,3,2,3,
        8,3,2,3,2,3,2,3,
        6,3,2,3,2,3,2,3,
        8,3,2,3,2,3,2,3,
        2,3,2,5,4,5,4,3,2,3,
        6,3,2,3,2,3,2,3,
```

```
    8,3,2,3,2,3,2,3,
    6,3,2,3,2,3,2,3,
    8,3,2,3,2,3,2,3,
    6,3,2,3,2,3,2,3,
    8,3,2,3,2,3,2,3,
    2,3,2,3,6,7,6,7,6,7,
    2,3,2,3,8,9,8,9,8,9,
    2,3,2,3,4,5,4,5,4,5,
    2,3,2,3,2,3,2,3,2,3,0};

private final static int[][] LINE = { // ライン
    {3,2,3,2,3,2,3},  // 0:端
    {3,1,1,1,1,1,3},  // 1:ゴール
    {3,0,0,0,0,0,3},  // 2:空(木あり)
    {2,0,0,0,0,0,2},  // 3:空(木なし)
    {2,3,0,0,0,3,2},  // 4:中(木あり)
    {2,2,0,0,0,2,2},  // 5:中(木なし)
    {2,2,2,3,0,0,3},  // 6:左(木あり)
    {2,2,2,2,0,0,2},  // 7:左(木なし)
    {3,0,0,3,2,2,2},  // 8:右(木なし)
    {2,0,0,2,2,2,2}}; // 9:右(木なし)

// 頂点
private final static int[] GROUND_QV = new int[]{ // 地面 - 四角形
     -260,0,-260, -260,0,260, 260,0,260, 260,0,-260};
private final static int[] TREE_TV = new int[]{   // 木 - 三角形
    0,400,0,  200,100,-100,  200,100, 100,
    0,400,0,  100,100,-200,  200,100,-100,
    0,400,0, -100,100,-200,  100,100,-200,
    0,400,0, -200,100,-100, -100,100,-200,
    0,400,0, -200,100, 100, -200,100,-100,
    0,400,0, -100,100, 200, -200,100, 100,
    0,400,0,  100,100, 200, -100,100, 200,
    0,400,0,  200,100, 100,  100,100, 200};
private final static int[] TREE_QV = new int[]{   // 木 - 四角形
     60,100, 60,  60,100,-60,  60,0,-60,  60,0, 60,
     60,100,-60, -60,100,-60, -60,0,-60,  60,0,-60,
    -60,100,-60, -60,100, 60, -60,0, 60, -60,0,-60,
    -60,100, 60,  60,100, 60,  60,0, 60, -60,0, 60};

// システム
private Graphics3D g3; // グラフィックス3D
private Transform trans    = new Transform(); // 視野座標
private Vector3D position = new Vector3D(0,300,1000); // 視点
private Vector3D look     = new Vector3D(0,300,0);    // 参照点
private Vector3D up       = new Vector3D(0,1,0);      // UPベクトル

private Primitive tipRoad;  // 道路
```

Next page. ⬇

```
    private Primitive tipGoal;  // ゴール
    private Primitive tipTreeT; // 木の三角形
    private Primitive tipTreeQ; // 木の四角形
    private Transform origin = new Transform(); // 原点

    // 初期化
    public void init(Graphics g) {
        int i;
        int[] arr;
        Vector3D vec;
        Light light;

        // グラフィックス3D
        g3 = (Graphics3D)g;

        // 視点座標の指定
        trans.lookAt(position,look,up);
        g3.setTransform(trans);
        g3.setPerspectiveView(10,5000,90);

        // 道路
        tipRoad = new Primitive(
            Primitive.PRIMITIVE_QUADS,
            Primitive.COLOR_PER_PRIMITIVE|
            Primitive.NORMAL_NONE,1);
        arr = tipRoad.getVertexArray();
        for (i=0;i<GROUND_QV.length;i++) {
            arr[i] = GROUND_QV[i];
        }
        tipRoad.getColorArray()[0] = (180<<16|180<<8|180);

        // ゴール
        tipGoal = new Primitive(
            Primitive.PRIMITIVE_QUADS,
            Primitive.COLOR_PER_PRIMITIVE|
            Primitive.NORMAL_NONE,1);
        arr = tipGoal.getVertexArray();
        for (i=0;i<GROUND_QV.length;i++) {
            arr[i] = GROUND_QV[i];
        }
        tipGoal.getColorArray()[0] = (255<<16|0<<8|0);

        // 木の三角形
        tipTreeT = new Primitive(
            Primitive.PRIMITIVE_TRIANGLES,
            Primitive.COLOR_PER_PRIMITIVE|
            Primitive.NORMAL_PER_FACE,8);
        arr = tipTreeT.getVertexArray();
        for (i=0;i<TREE_TV.length;i++) {
```

```
            arr[i] = TREE_TV[i];
        }
        tipTreeT.getColorArray()[0]=(27<<16|198<<8|137);
        arr = tipTreeT.getNormalArray();
        for (i=0;i<8;i++) {
            vec = new Vector3D(FastMath.cos(45*i),0,-(int)FastMath.sin(45*i));
            vec.normalize();
            arr[i*3  ] = (int)vec.getX()*4096;
            arr[i*3+1] = (int)vec.getY()*4096;
            arr[i*3+2] = (int)vec.getZ()*4096;

        }

        // 木の四角形
        tipTreeQ = new Primitive(
            Primitive.PRIMITIVE_QUADS,
            Primitive.COLOR_PER_PRIMITIVE|
            Primitive.NORMAL_PER_FACE,4);
        arr = tipTreeQ.getVertexArray();
        for (i=0;i<TREE_QV.length;i++) {
            arr[i] = TREE_QV[i];
        }
        tipTreeQ.getColorArray()[0] = (253<<16|198<<8|137);
        arr = tipTreeQ.getNormalArray();
        for (i=0;i<4;i++) {
            vec = new Vector3D(FastMath.cos(90*i),0,-(int)FastMath.sin(90*i));
            vec.normalize();
            arr[i*3  ] = (int)vec.getX()*4096;
            arr[i*3+1] = (int)vec.getY()*4096;
            arr[i*3+2] = (int)vec.getZ()*4096;
        }

        // 環境光源
        light = new Light();
        light.setMode(Light.AMBIENT);
        light.setIntensity(0.9f);
        g3.addLight(light,null);

        // 平行光源
        light = new Light();
        light.setMode(Light.DIRECTIONAL);
        light.setIntensity(0.5f);
        light.setVector(new Vector3D(100,0,100));
        g3.addLight(light,null);
    }

    // チップの取得
    public int getTip(int x,int z) {
        return LINE[MAP[(z+250)/500]][(x+250)/500];
```

Next page. ⬇

chapter

1
2
3
4
5
6
7
8
a

```
    }

    // マップの描画 ...❷
    public void draw(int x,int z,int dir) {
        int i,j;
        int a0,a1;

        // 視点座標の指定
        position.setX(x);
        position.setZ(z);
        look.setX(position.getX()-FastMath.cos(dir)*1400);
        look.setZ(position.getZ()-FastMath.sin(dir)*1400);
        trans.lookAt(position,look,up);
        g3.setTransform(trans);

        // 描画範囲の制限 ...❸
        a0 = z/500-10;
        a1 = z/500+6;
        if (dir>2048) {
            a0 = z/500-6;
            a1 = z/500+10;
        }
        if (a0<0)          a0 = 0;
        if (a1>MAP.length) a1 = MAP.length;

        // プリミティブの描画
        for (j=a0;j<a1;j++) {
            for (i=0;i<7;i++) {
                // 位置
                origin.setIdentity();
                origin.translate(i*500,0,j*500);

                // 道路
                if (LINE[MAP[j]][i]==0) {
                    g3.renderObject3D(tipRoad,origin);
                }
                // ゴール
                else if (LINE[MAP[j]][i]==1) {
                    g3.renderObject3D(tipGoal,origin);
                }
                // 木
                else if (LINE[MAP[j]][i]==3) {
                    g3.renderObject3D(tipTreeT,origin);
                    g3.renderObject3D(tipTreeQ,origin);
                }
            }
        }
        g3.flushBuffer();
    }
```

```
  }
```


DoJa Map.java

ⓐ

```
import com.docomostar.ui.graphics3d.Graphics3D;
import com.docomostar.ui.graphics3d.Light;
import com.docomostar.ui.graphics3d.Primitive;
import com.docomostar.ui.util3d.FastMath;
import com.docomostar.ui.util3d.Transform;
import com.docomostar.ui.util3d.Vector3D;
import com.docomostar.ui.Graphics;
import javax.microedition.io.Connector;
```

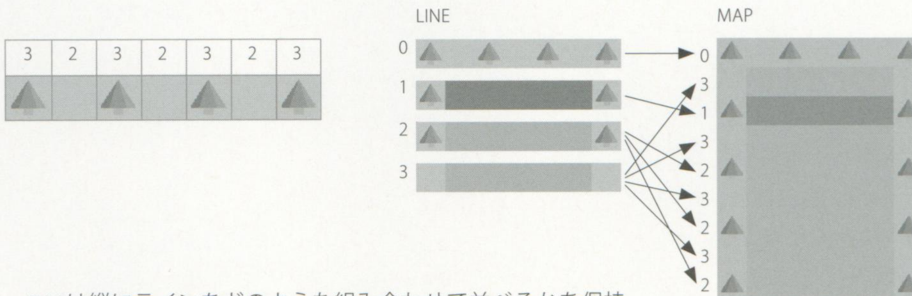
Map …❶：マップデータ

マップデータはint[][]型の変数「LINE」と、int[]型の変数「MAP」で構成されています。

LINEは横１行にマップチップをどのような組み合わせで並べるかを保持します。配列の要素は、以下です。

0	道路	2	緑
1	ゴール	3	木

つまり、{3,2,3,2,3,2,3}のような並びの時は、木、緑、木…　という並びなので、次のようになります。このゲームでは10のラインを用意しています。



MAPは縦にラインをどのような組み合わせで並べるかを保持します。配列の要素はLINEのインデックスです。同じラインが多数発生するようなマップの場合には、このようなデータ構造を使うとデータ量が少なくてすみます。

Map ...❷：マップの描画

マップの描画はMapクラスのdraw()メソッドで行います。引数でXZ座標と視点の方向を指定しています。

［Mapクラス］

void draw(*x, z, dir*)

［解説］ マップの描画

［引数］ *x* X座標：int型

z Z座標：int型

dir 方向：int型

Map ...❸：描画範囲の制限

マップデータ全ての描画を行うと処理が重たくなってしまうので、進行方向に10チップ分、後ろ方向に6チップ分のみ描画するように制限しています。

▶ADFの設定

ADFはAppNameとDrawAreaを次のように設定してください。

AppName	ポリゴンレーシング
DrawArea	240x200

付録：appendix

a

a-1 Star-1.0/DoJa-5.1対応端末 iアプリスペック一覧

本書で解説しているStar-1.0/DoJa-5.1プロファイルに対応した機種をシリーズごとに並べ直しました。

表では各機種におけるiアプリ対応プロファイル、iアプリの描画領域（横×縦　ドット）、フルアプリおよびウィジェットアプリのヒープ領域、デフォルトフォントのサイズについてまとめています。

- **対応プロファイル**は基本的にStar-1.0/DoJa-5.1プロファイル対応端末の「機種名」を掲載しましたが、Aシリーズをすべて掲載するために、DoJa-5.0LE、DoJa-4.1LE対応の端末を一部掲載しています。
- **最大iアプリサイズ**は"JARファイル+スクラッチパッド"の総容量を記載していますが、可変サイズ非対応の場合は"JARファイル／スクラッチパッド"で記載しています。
- **ヒープ容量**は、Javaヒープとネイティブデータヒープƒの区別がある場合は"Javaヒープ／ネイティブデータヒープ"が記載されており、区別がない場合は「Javaヒープ」です。ヒープについて詳細は089ページを参照してください。
- **描画領域**は、Panelクラス、Canvasクラスともに同じ領域が使えます。

シリーズ	機種名	対応プロファイル	最大iアプリサイズ（KB）	ヒープ容量（KB）フルアプリ（Java／ネイティブデータ）	ヒープ容量（KB）ウィジェットアプリ	描画領域（横×縦ドット）	デフォルトフォント（横×縦ドット）
A	F-01A	Star-1.0	2,048	23,552／8,192	2,503	480×864	24×24
	F-02A	DoJa-5.1LE	1,024	8,192／6,144	—	480×854	12×12
	F-03A	Star-1.0	2,048	23,552／8,192	2,366	480×960	24×24
	F-04A	DoJa-5.1LE	1,024	8,192／6,144	—	480×854	12×12
	F-05A（キッズケータイ）	DoJa-5.0LE	1,024	6,144／6,144	—	240×432	12×12
	F-06A	DoJa-5.1	1,024	6,144／6,144	—	480×864	12×12
	L-01A	DoJa-4.1LE	100/400	4,500／3,500	—	240×240	12×12
	N-01A	Star-1.0	2,048	32,768	3,527	480×854	24×24
	N-02A	Star-1.0	2,048	32,768	3,527	480×854	24×24
	N-03A	DoJa-5.1	1,024	13,312	—	240×427	12×12
	N-04A	Star-1.0	2,048	32,768	3,527	480×854	24×24
	P-01A	Star-1.0	2,048	20,480／4,608	1,638	480×854	24×24
	P-02A	Star-1.0	2,048	20,480／4,608	1,638	480×854	24×24
	P-03A	DoJa-5.1	1,024	15,360	—	240×426	12×12
	P-04A	DoJa-5.1	1,024	15,360	—	240×426	12×12

A	P-05A	DoJa-5.1	1,024	15,360	―	240×426	12×12
	SH-01A	Star-1.0	2,048	16,384／23,040	1,600	480×854	24×24
	SH-02A	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SH-03A	Star-1.0	2,048	16,384／23,040	1,600	480×854	24×24
	SH-04A	Star-1.0	2,048	16,384／23,040	1,600	480×854	24×24
906i	F906i	DoJa-5.1	1,024	6,144／ 6,144	―	480×864	12×12
	N906i	DoJa-5.1	1,024	13,312	―	480×854	12×12
	N906iL	DoJa-5.1	1,024	13,312	―	480×854	12×12
	N906iμ	DoJa-5.1	1,024	13,312	―	480×854	12×12
	P906i	DoJa-5.1	1,024	15,360	―	480×854	12×12
	SH906i	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SH906iTV	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SO906i	DoJa-5.1	1,024	16,384	―	480×864	12×12
905i	D905i	DoJa-5.1	1,024	6,144／6,144	―	480×864	12×12
	F905i	DoJa-5.1	1,024	6,144／6,144	―	480×864	12×12
	F905iBiz	DoJa-5.1	1,024	6,144／6,144	―	480×864	12×12
	N905i	DoJa-5.1	1,024	13,312	―	480×854	12×12
	N905iBiz	DoJa-5.1	1,024	13,312	―	480×854	12×12
	N905iμ	DoJa-5.1	1,024	13,312	―	480×854	12×12
	P905i	DoJa-5.1	1,024	15,360	―	480×854	12×12
	P905iTV	DoJa-5.1	1,024	15,360	―	480×854	12×12
	SH905i	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SH905iTV	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SO905i	DoJa-5.1	1,024	7,168／10,240	―	480×864	12×12
	SO905iCS	DoJa-5.1	1,024	7,168／10,240	―	480×864	12×12
80Xi	F884i（らくらくホン プレミアム）	DoJa-5.1LE	1,024	6,144／6,144	―	240×364	12×12
	F884iES（らくらくホンV）	DoJa-5.1LE	1,024	6,144／6,144	―	240×282	12×12
706i	F706i	DoJa-5.1LE	1,024	6,144／6,144	―	240×432	12×12
	N706i	DoJa-5.1	1,024	13,312	―	240×427	12×12
	N706ie	DoJa-5.1	1,024	13,312	―	240×427	12×12
	N706iII	DoJa-5.1	1,024	13,312	―	240×427	12×12
	P706ie	DoJa-5.1LE	1,024	15,360	―	240×426	12×12
	P706iμ	DoJa-5.1LE	1,024	15,360	―	240×426	12×12
	SH706i	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SH706iw	DoJa-5.1	1,024	15,360／10,240	―	480×854	12×12
	SO706i	DoJa-5.1	1,024	13,312	―	240×427	12×12

705i	N705i	DoJa-5.1	1,024	13,312	—	240×427	12×12
	N705iμ	DoJa-5.1	1,024	13,312	—	240×427	12×12
	P705i	DoJa-5.1LE	1,024	15,360	—	240×426	12×12
	P705iCL (PROSOLID μ)	DoJa-5.1LE	1,024	15,360	—	240×426	12×12
	P705iμ	DoJa-5.1LE	1,024	15,360	—	240×426	12×12
	SO705i	DoJa-5.1LE	1,024	13,312	—	240×427	12×12

a-2 クラス一覧：class index

本書で解説のあったクラス、メソッドをパッケージごとに並べ直し掲載しました。

※パッケージ、クラス/インタフェースの順に並べ、クラスの持つ定数やメソッド名とその機能、および解説のあるページ（表の右端）を記載しています。

Star-1.0 基本API

▶com.docomostarパッケージ

iアプリの基底、管理クラスを定義。

[FelicaAdhocEventクラス]：アドホック通信に関するイベントクラス

Hashtable **getReceivedParameter()**	パラメータの取得	222

[StarApplicationクラス]：Star iアプリの土台となるクラス

定数　280

void **activated(***activateInfo***)**	iアプリの開始時と再開時に呼ばれる	319
void **addEventListener(***event, listener***)**	イベントリスナの追加	222
void **started(***launchType***)**	iアプリ起動時に呼ばれる	057
void **updateStarApplication(***event***)**	イベント発生時に呼ばれる	222

[StarApplicationManagerクラス]：ADF（.jamファイル）を表すクラス

String **getSourceURL()**	ダウンロード元URLの取得	167
void **upgrade()**	iアプリのバージョンアップ	169

▶com.docomostar.deviceパッケージ

デバイス制御を行うクラスを定義。

[Cameraクラス]：携帯電話のカメラ機能に関するクラス

定数　146

void **disposeImages()**	撮影画像の破棄	147
static Camera **getCamera(*id*)**	Cameraオブジェクトの取得	146
Image **getImage(*index*)**	イメージの取得	147
InputStream **getInputStream(*index*)**	イメージの取得	147
int **getNumberOfImage()**	Cameraオブジェクトが保持している撮影画像数の取得	147
void **setAttribute(*attr*, *value*)**	属性の指定	146
void **setImageSize(*width*, *height*)**	写真の画像サイズの指定	146

[CodeReaderクラス]：コード認識機能に関するクラス

定数　155, 158

byte[] **getByte()**	コード認識結果の取得	156
static CodeReader **getCodeReader(*id*)**	コードリーダオブジェクトの取得	155
int **getResultType()**	コードの読み取り結果種別の取得	156
String **getString()**	コード認識結果の取得	156
void **read()**	コードの読み取り	156
void **setCode(*type*)**	読み取るコード種別の指定	155

[StorageDeviceクラス]：ストレージデバイスに関するクラス

Folder **getFolder(*accessToken*)**	フォルダの取得	198
static StorageDevice **getInstance(*deviceName*)**	SDカードと接続	197

▶com.docomostar.device.felicaパッケージ

FeliCa機能を利用するクラスを定義。

[AdhocDataTransferクラス]：アドホック通信による転送機能を制御するクラス

void **setup(*adfURL*, *command*, *params*)**	アドホック通信による連続データ転送の接続	228
void **terminateAdhoc()**	アドホック通信による連続データ転送の終了	231

[Felicaクラス]：FeliCa機能を制御するクラス

static void **close()**	FeliCaチップのクローズ	231
static AdhocDataTransfer **getAdhocDataTransfer()**	AdhocDataTransferオブジェクトの取得	228
static void **open()**	FeliCaチップのオープン	228
static void **turnOffRFPower()**	搬送波の出力の停止	231

▶com.docomostar.fsパッケージ

ストレージデバイス上のファイルシステムに関するクラスを定義。

[EncryptionAttributeクラス]：ファイルの暗号化に関するクラス

void **setEncryption(*encryption*)**	暗号化のON/OFFの指定	201

[Fileクラス]：ファイルを定義するクラス

定数　199

FileEntity **open(*mode*)**	ファイルとの接続	199

[Folderクラス]：フォルダを定義するクラス

File **createFile(*fileName*)**	ファイルの生成	198
File **createFile(*fileName*, *attributes*)**	ファイルの生成	200
File **getFile(*fileName*)**	ファイルの取得	198

[StarAccessTokenクラス]：iアプリからストレージデバイスやフォルダ・ファイルへのアクセス権を定義するクラス

定数　197

[StarStorageServiceクラス]：Star用ストレージデバイスサービスを定義するクラス

定数　197

static StarAccessToken **getAccessToken(*access*, *share*)**	StarAccessToken オブジェクトの生成	197

▶com.docomostar.fs.sdパッケージ

SDカードに関するクラスを定義。

[SDBindingEncryptionAttributeクラス]：暗号化を表すファイル属性を定義するクラス

void **setBlockSize**(*blockSize*)	暗号化ブロックサイズの指定	201

▶com.docomostar.ioパッケージ

入出力機能に関するクラスを定義。

[FelicaClientObexConnectionクラス]：アドホック通信による接続を定義するクラス

void **close**()	OBEXの切断	231
int **getResponseCode**()	レスポンスコードの取得	230
OutputStream **openOutputStream**()	出力ストリームの取得	230
void **sendRequest**()	リクエストを送信してレスポンスの受信を完了するまで待つ	230
void **setName**(*name*)	名前の指定	229
void **setOperation**(*operation*)	リクエストの種類の指定	229

[FileEntityクラス]：ファイルの実体を定義するクラス

InputStream **openInputStream**()	入力ストリームの取得	200
OutputStream **openOutputStream**()	出力ストリームの取得	199

[ObexConnectionインタフェース]：OBEXでのネットワークへの接続を定義するインタフェース

定数　229

[ServerObexConnectionインタフェース]：OBEX通信の接続に関するクラス

int **getContentLength**()	データの長さの取得	233
String **getName**()	名前の取得	232
int **getOperation**()	リクエストの種類の取得	232
InputStream **openInputStream**()	入力ストリームの取得	233
void **sendResponse**(*response*)	レスポンスを返す	233

▶com.docomostar.mediaパッケージ

イメージやサウンドを管理するクラス/インタフェースを定義。

[MediaImageインタフェース]：メディアイメージを定義するクラス

Image **getImage()**	イメージオブジェクトの取得	085
void **use()**	イメージの利用可能化	085

[MediaManagerクラス]：メディアデータ管理を定義するクラス

static MediaImage **getImage(***data***)**	イメージオブジェクト（バイト配列指定）取得	208
static MediaImage **getImage(***location***)**	イメージファイルの読み込み（場所指定）	085
static MediaSound **getSound(***data***)**	サウンドオブジェクト（バイト配列指定）取得	208
static MediaSound **getSound(***location***)**	サウンドファイルの読み込み（場所指定）	116

[MediaSoundインタフェース]：メディアサウンドを定義するクラス

void **dispose()**	メディアリソースの破棄	120
void **use()**	サウンドデータの有効化	116

▶com.docomostar.systemパッケージ

電話帳やメールなど、携帯電話ネイティブ機能を利用するクラスを定義

［ApplicationStoreクラス］：携帯電話のiアプリ管理機能にアクセスするクラス

int **getId()**	エントリIDの取得	169
static ApplicationStore **selectEntry()**	端末内のiアプリを選択	169

［Bookmarkクラス］：携帯電話のブックマーク管理機能にアクセスするクラス

static void **addEntry(***url, title***)**	ブックマークの登録	141

［ImageStoreクラス］：携帯電話のイメージデータ管理機能にアクセスするクラス

static int **addEntry(***mediaImage***)**	イメージデータを端末に保存	145
ImageStore **getImage()**	イメージの読み込み	145
InputStream **getInputStream()**	イメージの読み込み	145
static ImageStore **selectEntry()**	イメージデータを端末のデータフォルダから選択	145

［Launcherクラス］：iアプリから他のアプリケーションを起動するクラス

String[] **launch(***target, args***)**	ブラウザ起動	166

［Phoneクラス］：携帯電話の通話機能にアクセスするクラス
定数　130

static void **call(***phoneNumber***)**	音声通話	140

［PhoneBookクラス］：携帯電話の電話帳管理機能にアクセスするクラス

static int[] **addEntry(***name, kana, phoneNumbers, mailAddress, groupId***)**	電話帳エントリの新規登録	141

［PhoneBookGroupクラス］：携帯電話の電話帳グループ管理機能にアクセスするクラス

static void **addEntry(***name***)**	電話帳グループの新規登録	140

［PhoneSystemクラス］：携帯電話の基本機能を提供するクラス
定数　129, 131, 148

static int **getAttribute(***attr***)**	端末情報の取得	130
static int **getProperty(***attr***)**	個体識別情報の取得	129
static void **setAttribute(***attr, value***)**	属性の指定	148

[Scheduleクラス]：携帯電話のスケジューラ機能にアクセスするクラス

static boolean **addEntry(*param*)**	スケジュールの登録	141

[ScheduleParamクラス]：スケジュールデータを作成・登録するクラス

void **setAlarm(*flag*)**	アラームの有無の指定	142
void **setDate(*date*)**	日付の指定	142
void **setDescription(*description*)**	説明の指定	142

▶com.docomostar.uiパッケージ

ユーザインタフェイスの作成やイメージ描画のためのクラスを定義。

[AudioPresenterクラス]：サウンドデータの再生を定義するクラス

定数　118，119

static AudioPresenter **getAudioPresenter(*track*)**	オーディオプレゼンタの生成	117
void **pause()**	サウンドの一時停止	118
void **play()**	サウンドの再生	118
void **restart()**	サウンドの再開	119
void **setAttribute(*attr, value*)**	サウンドエフェクトの指定	119
void **setMediaListener(*listener*)**	メディアリスナの指定	117
void **setSound(*sound*)**	サウンドデータの指定	117
void **stop()**	サウンドの停止	118

[Canvasクラス]：キャンバスに関するクラス

定数　161

Graphics **getGraphics()**	Graphicsオブジェクトの取得	093
int **getHeight()**	キャンバスの高さの取得	067
int **getKeypadState()**	キー状態の取得	107
int **getWidth()**	キャンバスの幅の取得	067
void **imeOn(*text, displayMode, inputMode*)**	IMEの呼び出し	160
void **paint(*g*)**	描画時に呼ばれる	059
void **processEvent(*type, param*)**	キャンバスでのイベント発生時に呼ばれる	102，294
void **processIMEEvent(*type, text*)**	IMEイベント発生時に呼ばれる	161
void **setSoftLabel(*key, caption*)**	ソフトラベルの指定	102

[Displayクラス]：スクリーンやキーパッドの情報を取得するクラス
定数　102，103，108、294

static void **setCurrent(***frame***)**	実画面に表示する画面を指定	058

[Fontクラス]：文字の種別を定義するクラス
定数　067，068

static Font **getFont(***type, size***)**	フォントの取得	067
int **getHeight()**	文字列の高さの取得	069
int **stringWidth(***str***)**	文字列の幅の取得	069

[Graphicsクラス]：CanvasおよびImageに描画するグラフィックスクラス
定数　065，088

void **drawArc(***x, y, width, height, startAngle, arcAngle***)**	円弧の描画	077
void **drawImage(***image, x, y***)**	イメージの描画	086
void **drawLine(***x0, y0, x1, y1***)**	ラインの描画	074
void **drawPolyline(***xPoints, yPoints, nPoints***)**	ポリラインの描画	076
void **drawRect(***x, y, width, height***)**	四角形の描画	075
void **drawScaledImage(***image, dx, dy, width, height, sx, sy, swidth, sheight***)**	イメージの拡大縮小	087
void **drawString(***text, x, y***)**	文字列を描画	060
void **fillArc(***x, y, width, height, startAngle, arcAngle***)**	円弧の塗り潰し	078
void **fillPolygon(***xPoints, yPoints, nPoints***)**	ポリゴンの描画	077
void **fillRect(x, y, width, height)**	四角形の塗り潰し	066，075
static int **getColorOfName(***color***)**	色値の取得	065
static int **getColorOfRGB(***red, green, blue***)**	色値の取得	065
void **lock()**	ロック	094
void **setColor(***color***)**	色の指定	064
void **setFlipMode(***flipmode***)**	イメージの反転	088
void **setFont(***font***)**	フォントの指定	069
void **setPictoColorEnabled(***flag***)**	指定色で描画するか指定	070
void **setPixel(***x, y***)**	ピクセルの描画	073
void **setPixel(***x, y, color***)**	ピクセルの描画	074
void **unlock(***forced***)**	アンロック	094

[Imageクラス]：イメージを定義するクラス

void **dispose()**	イメージの破棄	089

[MediaListenerインタフェース]：メディアリスナイベントを定義するクラス

void **mediaAction(***source, type, param***)**	メディアリスナイベントの発生時に呼ばれる	118

▶com.docomostar.ui.flashplayerパッケージ

Flashをiアプリ上で再生するためのクラスを定義。

[FlashPlayerPaneクラス]：FlashをStar iアプリ上で操作するクラス

void **play()**	Flashコンテンツの再生	290
void **set(*data*)**	Flashコンテンツを指定	289
void **set(*data*, *rgb*)**	Flashコンテンツを指定	290

▶com.docomostar.ui.graphics3dパッケージ

3Dグラフィックスに関するクラス/インタフェースを定義。

[ActionTableクラス]：モデルの動きを表すクラス

int **getMaxFrame(*index*)**	最大フレーム数の取得	256

[DrawableObject3Dクラス]：描画可能な3Dオブジェクトを定義するクラス

定数　255

void **setPerspectiveCorrectionEnabled(*flag*)**	テクスチャの歪み補正の有効化/無効化を指定	247

[Figureクラス]：モデルの形状データを保持するクラス

void **setBlendMode(*mode*)**	ブレンドモードの指定	255
void **setTime(*frame*)**	アクションのフレーム指定	256
void **setTransparency(*rate*)**	透明度をパーセントで設定	255

[Graphics3Dインタフェース]：3Dグラフィックスを提供するインタフェース

void **addLight(*light*, *trans*)**	光源の追加	250
void **flushBuffer()**	レンダリング結果の画面への反映	256
void **renderObject3D(*obj3d*, *trans*)**	3Dオブジェクトのレンダリング	256
void **resetLights()**	光源の解除	250
void **setParallelView(*width*, *height*)**	平行投影の指定	273
void **setPerspectiveView(*zNear*, *zFar*, *angle*)**	透視投影の指定	274
void **setPerspectiveView(*zNear*, *zFar*, *width*, *height*)**	透視投影の指定	274

chapter
1
2
3
4
5
6
7
8
a

[Groupクラス]：3Dオブジェクトグループを表すクラス

Object3D **getElement**(*index*)	3Dオブジェクトの取得	247

[Lightクラス]：光源データを保持するクラス

定数　250

int **getMaxLights**()	光源の最大数を取得	250
void **setAttenuation**(*constant, linear, quadratic*)	光源の減衰に関するパラメータの指定	253
void **setColor**(*color*)	光源の色の指定	251
void **setIntensity**(*intensity*)	光源の強度の指定	251
void **setMode**(*mode*)	光源の種類の指定	250
void **setPosition**(*v*)	光源の位置の指定	253
void **setSpotAngle**(*angle*)	スポット光源の角度の指定	254
void **setSpotExponent**(*exponent*)	スポット光源モード用の輝度分布の指定	254
void **setVector**(*v*)	光源の向きの指定	252

[Object3Dクラス]： 3Dオブジェクトの基底となるクラス

static Object3D **createInstance**(*data*)	3Dオブジェクトの生成	246
static Object3D **createInstance**(*in*)	3Dオブジェクトの生成	247

[Primitiveクラス]：プリミティブ図形を描画するクラス

定数　262, 263, 264 ,265, 267

Primitive(*type, param, num*)	Primitiveクラスのコンストラクタ	262
int[] **getColorArray**()	色情報の取得	264
int[] **getNormalArray**()	法線情報の取得	264
int[] **getPointSpriteArray**()	ポイントスプライト情報の取得	266
int[] **getTextureCoordArray**()	テクスチャ情報の取得	265
int[] **getVertexArray**()	頂点情報の取得	263
void **setTexture**(*texture*)	テクスチャの指定	267

[Transformクラス]：三次元アフィン変換用の行列クラス

void **lookAt**(*position, look, up*)	視点から参照点を見た時の変換行列の計算	248
void **rotate**(*v, angle*)	変換行列の回転	249
void **rotate**(*x, y, z, angle*)	変換行列の回転	249
void **scale**(*v*)	変換行列の拡大縮小	249
void **scale**(*x, y, z*)	変換行列の拡大縮小	248

[TextBoxクラス]：テキスト入力コンポーネントを定義するクラス

定数　160

▶com.docomostar.ui.util3dパッケージ

3Dグラフィックスや3Dサウンドで使用するクラスを定義。

[FastMathクラス]：float型の高速な数値演算を提供するクラス

static float **cos**(*a*)	余弦（cos、コサイン）の計算	338
static float **sin**(*a*)	正弦（sin、サイン）の計算	338

▶com.docomostar.utilパッケージ

さまざまなユーティリティクラスを定義。

[JarInflaterクラス]：JARファイルからエントリ取り出すクラス

JarInflater(*in*)	JarInflaterクラスのコンストラクタ	217
void **close**()	切断を行う	218
InputStream **getInputStream**(*name*)	入力ストリームの取得	218

[ScheduleDateクラス]：スケジュール予定を指定するクラス

定数　143, 144

ScheduleDate(*type*, *repeatCount*, *spanSet*, *start*, *end*)	ScheduleDateクラスのコンストラクタ	143
ScheduleDate(*type*, *start*, *end*)	ScheduleDateクラスのコンストラクタ	142
void **set**(*attr*, *value*)	属性値の指定	144

▶com.docomostar.opt.uiパッケージ（Star-1.0オプションAPI）

拡張されたユーザインタフェースに関するクラスを定義。

[TouchDeviceクラス]：タッチパネルを利用するクラス

static int **getX**()	最終タッチのX座標の取得	294
static int **getY**()	最終タッチのY座標の取得	294
static void **setEnabled**( )	タッチパネルの有効無効を指定	293

DoJa-5.1基本API

▶com.nttdocomo.device.felicaパッケージ

FeliCaを利用するクラスを定義。

[FelicaAdhocListenerインタフェース]：アドホック通信によるデータ転送に関するクラス

boolean **requestReceived(***receivedParams***)**	アドホック通信の受信	223

▶com.nttdocomo.systemパッケージ

携帯電話のネイティブ機能を利用するクラスを定義。

[Scheduleクラス]：スケジューラ機能にアクセスするクラス

static boolean **addEntry(***description, date, alarm***)**	スケジュールの登録	143

▶com.nttdocomo.uiパッケージ

iアプリ/ユーザインタフェースの作成に関するクラスを定義。

[Fontクラス]：文字種別に関するクラス

static Font **getFont(***type***)**	フォントの取得	068

[IApplicationクラス]：iアプリの基本的なクラス

定数　300

int **getLaunchType()**	アプリケーションの起動形態の取得	300
String **getSourceURL()**	ダウンロード元URLの取得	168
String[] **launch(***target, args***)**	ブラウザ起動	166
void **resume()**	iアプリの再開時に呼ばれる	319
void **start()**	アプリケーション起動時に呼ばれる	057

[MApplicationクラス]：待ち受けiアプリに関するクラス

定数 298

void **deactivate()**	活性化状態から非活性化状態への遷移	300
int **getWakeupTimer()**	ウェイクアップイベントを発生するまでの時間の取得	299
void **processSystemEvent(***type, param***)**	システムイベント発生時に呼ばれる	298
void **resetWakeupTimer()**	ウェイクアップイベントの解除	299
void **setClockTick(***flag***)**	クロックイベントを発生させるかの指定	299
void **setWakeupTimer(***time***)**	ウェイクアップイベントを発生するまでの時間の指定	299
void **sleep()**	非活性化状態から休眠状態への遷移	300

▶com.nttdocomo.utilパッケージ

さまざまなユーティリティクラスを定義。

[Phoneクラス]：携帯電話のネイティブ機能にアクセスするためのクラス

static int **getProperty(***attr***)**	個体識別情報の取得	130

[ScheduleDateクラス]：スケジュールの時刻を指定するためのクラス

ScheduleDate(*type***)**	ScheduleDateクラスのコンストラクタ	144

J2ME CLDC 1.0.4

▶java.ioパッケージ

データストリームによる入出力に関するクラスを定義。

[InputStreamクラス]：バイト入力ストリームを表現するクラス

void **close()**	入力ストリームの切断	188
int **read()**	バイトデータの読み込み	188
byte[] **read(***data***)**	バイトデータの読み込み	188
void **read(***data, off, len***)**	バイトデータの読み込み	187

[OutputStreamクラス]：バイト出力ストリームを表現するクラス

void **close()**	出力ストリームの切断	186
void **write(***data***)**	バイトデータ（byte[]型)の書き込み	186
void **write(***data***)**	バイトデータ（int型)の書き込み	186
void **write(***data, off, len***)**	バイトデータの書き込み	185

▶java.langパッケージ

Javaプログラムにおける基本的なクラスを提供。

[Integerクラス]：int型に関するクラス

static void **parseInt(***str***)**	文字列の数値変換	329

[Stringクラス]：文字列を表すクラス

byte[] **getBytes()**	バイトデータの取得	186

[Threadクラス]：実行スレッドに関するクラス

static void **sleep(***time***)**	スリープ	094

▶java.utilパッケージ

さまざまなユーティリティクラスを定義。

[Calendarクラス]：日付の取得と設定を行うための抽象クラス

定数　144

▶javax.microedition.ioパッケージ

汎用接続用のクラスを定義。

[Connectorクラス]：接続に関するクラス

定数　229

static Connection **open(***name, mode, timeouts***)**	接続を行う	207, 229
static InputStream **openInputStream(***location***)**	入力ストリームの接続	187, 289
static OutputStream **openOutputStream(***location***)**	出力ストリームの接続	185

index

索引

記号・数字

A

B

C

index

G

index

か行

さ行

Star iアプリ開発テキストブック
Aシリーズ対応
2009年3月24日　初版第1刷発行

著者　布留川英一
発行者　中川信行
発行所　株式会社 毎日コミュニケーションズ
〒100-0003
東京都千代田区一ツ橋1-1-1
パレスサイドビル
TEL:　03-6267-4477（販売営業）
03-6267-4431（編集）

注文専用ダイヤル　048-485-6815

読者質問専用E-mail　pc-books@mycom.co.jp
URL　http://book.mycom.co.jp

ISBN978-4-8399-3077-6

デザイン　株式会社 ブランク
印刷・製本　図書印刷 株式会社

▶ 著者略歴
布留川 英一（ふるかわひでかず）
1975年群馬県出身。会津大学コンピュータ理工学部コンピュータソフトウェア学科卒。Javaプログラミングの連載「KVMゲームプログラミング」（『JAVA PRESS』誌、技術評論社刊）からライターとして活動を始める。2005年に独立し、ユビキタスエンターテインメントで次世代コンテンツを作成中。
ん・ぱか工房 http://npaka.net